Mobile Intel® Celeron® Processor at 466 MHz, 433 MHz, 400 MHz, 366 MHz, 333 MHz, 300 MHz, and 266 MHz Specification Update

Release Date: September 2002

Order Number: 244444-036

The Mobile Intel® Celeron® processor at 466 MHz, 433 MHz, 400 MHz, 366 MHz, 333 MHz, 300 MHz, and 266 MHz may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are documented in this Specification Update.

MOBILE INTEL® CELERON® PROCESSOR at 466 MHz, 433 MHz, 400 MHz, 366 MHz, 333 MHz, 300 MHz, and 266 MHz SPECIFICATION UPDATE

CONTENTS

REVISION HISTORY ii
PREFACE v
GENERAL INFORMATION 1
Mobile Intel® Celeron® Processor (Micro-PGA) Markings 1
Mobile Intel® Celeron® Processor (BGA) Markings 2
Intel® Celeron® Processor Mobile Module Markings 3
IDENTIFICATION INFORMATION 6
SUMMARY OF CHANGES 9
Summary of Errata 10
Summary of Documentation Changes 13
Summary of Specification Clarifications 14
Summary of Specification Changes 14
ERRATA 15
DOCUMENTATION CHANGES 51
SPECIFICATION CLARIFICATIONS 68
SPECIFICATION CHANGES 70

REVISION HISTORY

Date of Revision	Version	Description
January 1999	-001	This document is the first Specification Update for the Intel® Mobile Celeron® processor.
March 1999	-002	Updated the Documentation Changes and Specifications Clarifications sections. Changed S-Spec Definition.
April 1999	-003	Added Erratum H43. Updated Processor Identification Information table and added footnote. Updated Processor Mobile Module Information table. Added MMC-2 marking diagram.
May 1999	-004	Updated Processor Identification Information table. Added Erratum H44. Added Documentation Changes H2 through H6. Added Specification Clarifications H1 and H2. Added Specification Change H1. Updated BGA1 marking diagram.
June 1999	-005	Updated the Mobile Intel Celeron Processor Identification Information table. Added µPGA1 marking diagram. Added Erratum H45. Added Documentation Changes H7 through H10. Added Specification Change H2.
July 1999	-006	Added Erratum H46. Updated heading in Identification Information table. Updated the Mobile Intel Celeron Processor Identification and the Intel Celeron Processor Mobile Module Identification Information tables. Added Documentation Change H11. Added mcpA0 stepping to Summary Table of Changes. Updated Documentation Changes, Specification Clarifications, and Specification Changes introduction paragraphs.
August 1999	-007	Updated Preface, Documentation Changes, Specification Clarifications, and Specification Changes paragraphs. Added Documentation Change H12. Updated Codes Used in Summary Table. Updated column heading in Errata, Documentation Changes, Specification Clarifications and Specification Changes tables.
October 1999	-008	Added erratum H47
November 1999	-009	Added Errata H48, H49; added Documentation Change H13; added Specification Clarification H3; added Brand ID column in the *Identification Information* table. Updated the *Mobile Intel Celeron Processor Identification Information* table and notes. Updated the *Intel Celeron Processor Mobile Module Identification Information* table and notes. Added 433 MHz, and 466 MHz references.
December 1999	-010	Added Erratum H50 and Documentation Change H14
January 2000	-011	Added Erratum H51 and Documentation Change H15.
February 2000	-012	Updated Specification Update title to reflect speeds documented. Revised Erratum H49; Added Erratum H52; Added Document Change H16. Updated Summary of Changes product letter codes.

REVISION HISTORY

Date of Revision	Version	Description
March 2000	-013	Updated the *General Information* markings. Updated Preface document reference. Revised Erratum H48 and added Erratum H53.
April 2000	-014	Updated the Preface with new references; Updated the Intel Celeron Processor Mobile Module Markings section.
June 2000	-015	Added Erratum H54.
July 2000	-016	Added Erratum H55, H56.
August 2000	-017	Added Erratum H57.
September 2000	-018	Added Erratum H58, H59; Revised Erratum H34, H48, H52; Added Documentation Changes H17, H18
October 2000	-019	Updated the reference to the published documents in the Preface; Added Erratum H60; Added Documentation Changes H19, H20.
November 2000	-020	Added Erratum H61.
December 2000	-021	Updated Specification Update product key to include the Intel® Pentium® 4 processor, Revised Erratum H2; Added Documentation Changes H21 through H26.
January 2001	-022	Revised Erratum H2; Added Documentation Changes H27 and H28.
February 2001	-023	Revised Documentation Change H27 and Added H29.
March 2001	-024	Added Erratum H62 and H63.
May 2001	-025	Added Erratum H64
August 2001	-026	Updated Summary of Changes; Added Erratum H65; Added Documentation Change H30
October 2001	-027	Updated the Celeron® trademark to a registered trademark; Updated Summary of Changes; Updated Documentation Changes by removing old items that were incorporated in the new documents referenced in this spec update.
November 2001	-028	Updated Summary of Changes; Added Documentation Changes H2, H3, H4, H5, H6.
January 2002	-029	Updated Summary of Changes; Added Documentation Changes H7, H8, H9, H10, and H11.
March 2002	-030	Updated Summary of Changes; Added Erratum H66; Added Documentation Change H12.
April 2002	-031	Updated the Documentation Changes by removing old items that already have been incorporated in the published Software Developer's manual.
May 2002	-032	Added Documentation Change H2.
June 2002	-033	Updated Summary of Changes; Added Documentation Change H3 and H4; Added Erratum H67.

REVISION HISTORY

Date of Revision	Version	Description
July 2002	-034	Updated Summary of Changes; Removed old items that have been added to the Software Developers Manual; Added Documentation Change H4, H5, H6, H7, H8, H9, H10, H11, H12, and H13.
August 2002	-035	Updated the Documentation Changes summary section by removing old items that already have been incorporated in the published Software Developer's manual.
September 2002	-036	Updated the Documentation Changes summary section; Added Documentation Changes H4, H5, H6, H7, H8, H9, H10, H11, H12, H13, H14, H15, H16, H17, H18, H19, H20, H21, H22, H23, H24, and H25.

PREFACE

This document is an update to the specifications contained in the following documents:

- *P6 Family of Processors Hardware Developer's Manual* (Order Number 244001)
- *Intel® Mobile Celeron® Processor in Micro-PGA and BGA Package at 466 MHz, 433 MHz, 400 MHz, 366 MHz, 333 MHz, 300 MHz and 266 MHz* datasheet (Order Number 245112)
- *Intel® Celeron® Processor Mobile Module: Mobile Module Connector 1 (MMC-1) at 400 MHz, 366 MHz, 333 MHz, 300 MHz, and 266 MHz* datasheet (Order Number 2455426)
- *Intel® Celeron® Processor Mobile Module: Mobile Module Connector 2 (MMC-2) at 400 MHz, 366 MHz, 333 MHz, 300 MHz, and 266 MHz* datasheet (Order Number 245425)
- *Intel® Mobile Celeron® Processor: Mobile Module MMC-1 at 466 MHz and 433 MHz* datasheet (Order Number 245101)
- *Intel® Mobile Celeron® Processor: Mobile Module MMC-2 at 466 MHz and 433 MHz* datasheet (Order Number 245102)
- *Intel Architecture Software Developer's Manual*, *Volumes 1, 2 and 3* (Order Numbers 243190, 243191, and 243192, respectively)

It is intended for hardware system manufacturers and software developers of applications, operating systems, or tools. It contains S-Specs, Errata, Documentation Changes, Specification Clarifications and, Specification Changes.

Nomenclature

S-Spec Number is a five-digit code used to identify products. Products are differentiated by their unique characteristics, e.g., core speed, L2 cache size, package type, etc. as described in the processor identification information table. Care should be taken to read all notes associated with each S-Spec number.

Errata are design defects or errors. Errata may cause the Mobile Intel Celeron processor or the Mobile Intel Celeron Module's behavior to deviate from published specifications. Hardware and software designed to be used with any given processor must assume that all errata documented for that processor are present on all devices unless otherwise noted.

Documentation Changes include typos, errors, or omissions from the current published specifications. These changes will be incorporated in the next release of the specifications.

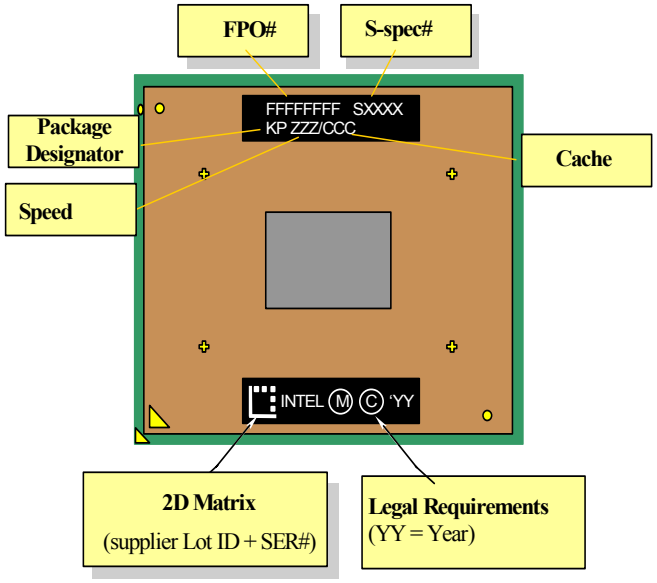
Specification Clarifications describe a specification in greater detail or further highlight a specification's impact to a complex design situation. These clarifications will be incorporated in the next release of the specifications.

Specification Changes are modifications to the current published specifications for the Intel® Mobile Celeron® processor or the Intel® Celeron® Processor Mobile Module. These changes will be incorporated in the next release of the specifications.

Specification Update for the Mobile Intel® Celeron® Processor at 466 MHz, 433 MHz, 400 MHz, 366 MHz, 333 MHz, 300 MHz, and 266 MHz

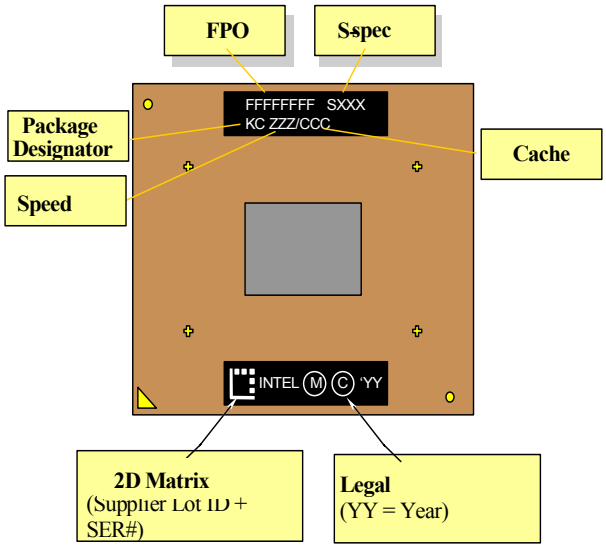
GENERAL INFORMATION

Mobile Intel® Celeron® Processor (Micro-PGA) Markings



Mobile Intel® Celeron® Processor in Micro-PGA Package

Mobile Intel® Celeron® Processor (BGA) Markings



Mobile Intel® Celeron® Processor in BGA1 Package

Intel® Celeron® Processor Mobile Module Markings

The Product Tracking Code (PTC) determines the Intel assembly level of the module. The PTC is on the secondary side of the module and provides the following information:

Example: ***PMG33302001AA***

- The PTC will consist of 13 characters as identified in the above example and can be broken down as follows:

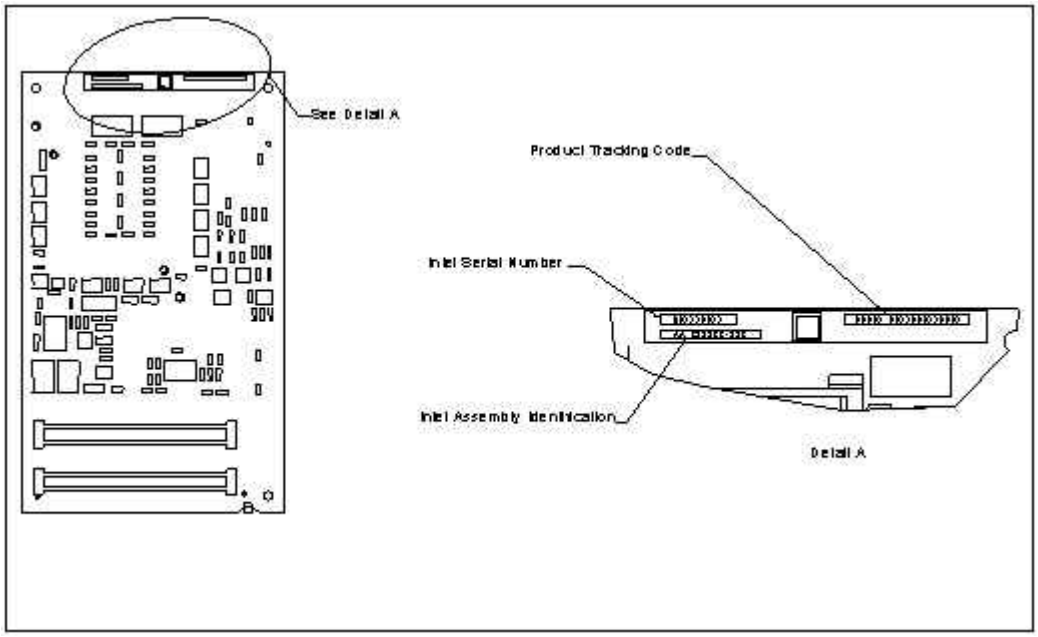
AABCCCDDEEEFF

- Definition:

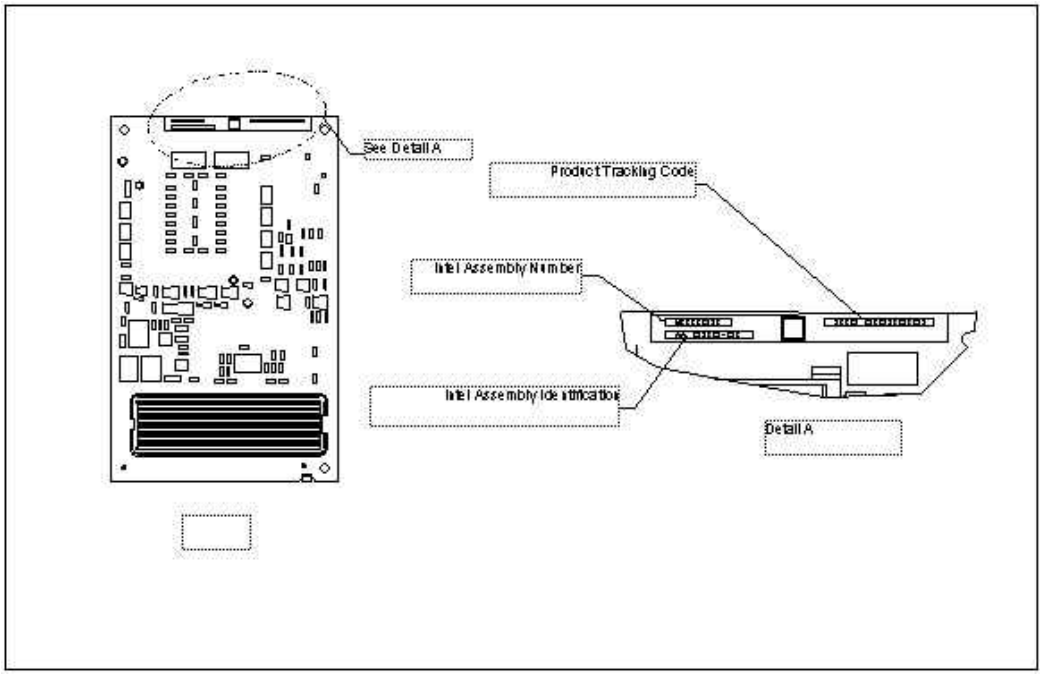
AA	-	Processor Module = PM
B	-	Celeron® Processor Mobile Module (MMC-1) = H Celeron® Processor Mobile Module (MMC-2) = I
CCC	-	Speed Identity = 466, 400, 366, 333, 300, or 266, etc.
DD	-	Cache Size = 01 (128KB)
EEE	-	Notifiable Design Revision (Start at 001)
FF	-	Notifiable Processor Revision (Start at AA)

- Note: For other Intel Mobile Modules, the second field (B) is defined as:

  Pentium® II Processor Mobile Module (MMC-1) = D
  Pentium® II Processor Mobile Module (MMC-2) = E
  Pentium® II Processor with On-die Cache Mobile Module (MMC-1) = F
  Pentium® II Processor with On-die Cache Mobile Module (MMC-2) = G



Intel® Celeron® Processor Mobile Module (MMC-1)



Intel® Celeron® Processor Mobile Module (MMC-2)

IDENTIFICATION INFORMATION

The Mobile Intel® Celeron® processor or the Intel® Celeron® Processor Mobile Module can be identified by the following values:

Family[1]	266, 300, 333, 366 , 400, 433, 466 MHz Model 6[2]	Brand ID
0110	0110	00h (not supported)

NOTES:

1. The Family corresponds to bits [11:8] of the EDX register after RESET, bits [11:8] of the EAX register after the CPUID instruction is executed with a 1 in the EAX register, and the generation field of the Device ID register accessible through Boundary Scan.
2. The Model corresponds to bits [7:4] of the EDX register after RESET, bits [7:4] of the EAX register after the CPUID instruction is executed with a 1 in the EAX register, and the model field of the Device ID register accessible through Boundary Scan.

The Mobile Intel Celeron processor and the Intel Celeron Processor Mobile Module's second level (L2) cache size can be determined by the following register contents:

128-Kbyte Unified L2 Cache[1]	41h

NOTES:

1 For the Intel® Mobile Celeron® processor and the Intel® Celeron® Processor Mobile Module, the unified L2 cache size will be returned as one of the cache/TLB descriptors when the CPUID instruction is executed with a 2 in the EAX register.

Mobile Intel® Celeron® Processor Identification Information

S-Spec	Product Steppings	CPUID	Speed (MHz) Core/Bus	Integrated L2 Size (Kbytes)	Package	Notes
SL23X	mcbA0	066Ah	233/66	128	BGA	1
SL23Y	mcbA0	066Ah	266/66	128	BGA	1
SL3AH	mcbA0	066Ah	300/66	128	BGA	1
SL3C8	mcbA0	066Ah	333/66	128	BGA	1
SL3C7	mcbA0	066Ah	366/66	128	BGA	1
SL3DQ	mcbA0	066Ah	266/66	128	BGA	2
SL3GQ	mcbA0	066Ah	400/66	128	BGA	1
SL3KA	mcbA0	066Ah	433/66	128	Micro-PGA	3
SL3KC	mcbA0	066Ah	466/66	128	Micro-PGA	3
SL3HM	mcpA0	066Ah	266/66	128	Micro-PGA	1
SL3HN	mcpA0	066Ah	300/66	128	Micro-PGA	1
SL3HP	mcpA0	066Ah	333/66	128	Micro-PGA	1
SL3HQ	mcpA0	066Ah	366/66	128	Micro-PGA	1
SL3GR	mcpA0	066Ah	400/66	128	Micro-PGA	1
SL3KB	mcpA0	066Ah	433/66	128	Micro-PGA	3
SL3KD	mcpA0	066Ah	466/66	128	Micro-PGA	3

NOTES:

1. VCC_CORE is specified for 1.6 V +/-135 mV for these Intel® Mobile Celeron® processors.
2. VCC_CORE is specified for 1.5 V +/-135 mV for these Mobile Intel Celeron processors.
3. VCC_CORE is specified for 1.9 V +/-135 mV for these Mobile Intel Celeron processors.

Intel® Celeron® Processor Mobile Module Identification Information

PTC	Product Steppings	CPUID	Speed (MHz) Core/Bus	Integrated L2 Size (Kbytes)	Package	Notes
PMH26601001AA	cmmA0	066Ah	266/66	128	MMC-1	1
PMH30001001AA	cmmA0	066Ah	300/66	128	MMC-1	1
PMH33301001AA	cmmA0	066Ah	333/66	128	MMC-1	1
PMH36601001AA	cmmA0	066Ah	366/66	128	MMC-1	1
PMH40001001AA	cmmA0	066Ah	400/66	128	MMC-1	1
PMH43301001AA	cmmA0	066Ah	433/66	128	MMC-1	2
PMH46601001AA	cmmA0	066Ah	466/66	128	MMC-1	2
PMI26601001AA	cmmA0	066Ah	266/66	128	MMC-2	1
PMI30001001AA	cmmA0	066Ah	300/66	128	MMC-2	1
PMI30001001AA	cmmA0	066Ah	333/66	128	MMC-2	1
PMI36601001AA	cmmA0	066Ah	366/66	128	MMC-2	1
PMI40001001AA	cmmA0	066Ah	400/66	128	MMC-2	1
PMI43301001AA	cmmA0	066Ah	433/66	128	MMC-2	2
PMI46601001AA	cmmA0	066Ah	466/66	128	MMC-2	2

NOTES:

1. Vcore voltage is 1.6V.
2. Vcore voltage is 1.9V.

SUMMARY OF CHANGES

The following table indicates the Errata, Documentation Changes, Specification Clarifications, or Specification Changes that apply to Mobile Intel® Celeron® processors. Intel intends to fix some of the errata in a future stepping of the component, and to account for the other outstanding issues through documentation or specification changes as noted. This table uses the following notations:

CODES USED IN SUMMARY TABLE

X:	Erratum, Documentation Change, Specification Clarification or Specification Change applies to the given processor stepping.
(No mark) or (blank box):	This item is fixed in or does not apply to the given stepping.
Fix:	This erratum is intended to be fixed in a future stepping of the component.
Fixed:	This erratum has been previously fixed.
NoFix:	There are no plans to fix this erratum.
Doc:	Intel intends to update the appropriate documentation in a future revision.
PKG:	This column refers to errata on the Intel® Mobile Celeron® processor or the Intel® Celeron® Processor Mobile Module substrate.
AP:	APIC related erratum.
Shaded:	This item is either new or modified from the previous version of the document.

Each Specification Update item is prefixed with a capital letter to distinguish the product. The key below details the letters that are used in Intel's microprocessor Specification Updates:

A = Intel® Pentium® II processor

B = Mobile Intel® Pentium® II processor

C = Intel® Celeron® processor

D = Intel® Pentium® II Xeon™ processor

E = Intel® Pentium® III processor

G = Intel® Pentium® III Xeon™ processor

H = Mobile Intel® Celeron® processor at 466 MHz, 433 MHz, 400 MHz, 366 MHz, 333 MHz, 300 MHz, and 266 MHz

K = Mobile Intel® Pentium® III processor

M = Mobile Intel® Celeron® processor at 500 MHz, 450 MHz, and 400A MHz

N = Intel® Pentium® 4 processor

P = Intel® Xeon™ processor

T = Mobile Intel® Pentium® 4 processor-M

V = Mobile Intel® Celeron® processor on .13 Micron Process in Micro-FCPGA Package

The Specification Updates for the Pentium® processor, Pentium® Pro processor, and other Intel products do not use this convention.

Summary of Errata

NO.	mcbA0	mcpA0	cmmA0	Plans	ERRATA
H1	X	X	X	NoFix	FP data operand pointer may be incorrectly calculated after FP access which wraps 64-Kbyte boundary in 16-bit code
H2	X	X	X	NoFix	Differences exist in debug exception reporting
H3	X	X	X	NoFix	Code fetch matching disabled debug register may cause debug exception
H4	X	X	X	NoFix	Double ECC error on read may result in BINIT#
H5	X	X	X	NoFix	FP inexact-result exception flag may not be set
H6	X	X	X	NoFix	BTM for SMI will contain incorrect FROM EIP
H7	X	X	X	NoFix	I/O restart in SMM may fail after simultaneous MCE
H8	X	X	X	NoFix	Branch traps do not function if BTMs are also enabled
H9	X	X	X	NoFix	Machine check exception handler may not always execute successfully
H10	X	X	X	NoFix	MCE due to L2 parity error gives L1 MCACOD.LL
H11	X	X	X	NoFix	LBER may be corrupted after some events
H12	X	X	X	NoFix	BTMs may be corrupted during simultaneous L1 cache line replacement
H13	X	X	X	Fix	Potential early deassertion of LOCK# during split-lock cycles
H14	X	X	X	NoFix	A20M# may be inverted after returning from SMM and Reset
H15	X	X	X	Fix	Reporting of floating-point exception may be delayed
H16	X	X	X	NoFix	Near CALL to ESP creates unexpected EIP address
H17	X	X	X	Fix	Built-in self test always gives nonzero result
H18	X	X	X	Fix	Cache state corruption in the presence of page A/D-bit setting and snoop traffic
H19	X	X	X	Fix	Snoop cycle generates spurious machine check exception
H20	X	X	X	Fix	MOVD/MOVQ instruction writes to memory prematurely
H21	X	X	X	NoFix	Memory type undefined for nonmemory operations
H22	X	X	X	NoFix	FP data operand pointer may not be zero after power on or reset

Summary of Errata

NO.	mcbA0	mcpA0	cmmA0	Plans	ERRATA
H23	X	X	X	NoFix	MOVD following zeroing instruction can cause incorrect result
H24	X	X	X	NoFix	Premature execution of a load operation prior to exception handler invocation
H25	X	X	X	NoFix	Read portion of RMW instruction may execute twice
H26	X	X	X	Fix	Intervening writeback may occur during locked transaction
H27	X	X	X	NoFix	MC2_STATUS MSR has model-specific error code and machine check architecture error code reversed
H28	X	X	X	NoFix	Mixed cacheability of lock variables is problematic in MP systems
H29			X	Fix	Thermal sensor may assert SMBALERT# incorrectly
H30	X	X	X	NoFix	MOV with debug register causes debug exception
H31	X	X	X	NoFix	Upper four PAT entries not usable with Mode B or Mode C paging
H32	X	X	X	Fix	Incorrect memory type may be used when MTRRs are disabled
H33	X	X	X	Fix	Misprediction in program flow may cause unexpected instruction execution
H34	X	X	X	NoFix	Data breakpoint exception in a displacement relative near call may corrupt eip
H35	X	X	X	NoFix	System bus ECC not functional with 2:1 ratio
H36	X	X	X	Fix	Fault on REP CMPS/SCAS operation may cause incorrect EIP
H37	X	X	X	NoFix	RDMSR and WRMSR to invalid MSR address may not cause GP fault
H38	X	X	X	NoFix	SYSENTER/SYSEXIT instructions can implicitly load “null segment selector” to SS and CS registers
H39	X	X	X	NoFix	PRELOAD followed by EXTEST does not load boundary scan data
H40	X	X	X	NoFix	Far jump to new TSS with D-bit cleared may cause system hang
H41	X	X	X	NoFix	Incorrect chunk ordering may prevent execution of the Machine Check Exception handler after BINIT#
H42	X	X	X	NoFix	Resume Flag may not be cleared after debug exception
H43			X	Fix	Processor may return invalid parameters on execution of the CPUID instruction

Summary of Errata

NO.	mcbA0	mcpA0	cmmA0	Plans	ERRATA
H44	X	X	X	NoFix	Internal cache protocol violation may cause system hang
H45	X	X	X	NoFix	GP# fault on WSMSR to ROB_CR_BKUPTMPDR6
H46	X	X	X	NoFix	Machine check exception may occur due to improper line eviction in the IFU
H47	X	X	X	NoFix	Lower Bits of SMRAM SMBASE Register Cannot Be Written With an ITP
H48	X	X	X	NoFix	Task switch may cause wrong PTE and PDE access bit to be set
H49	X	X	X	NoFix	Unsynchronized cross-modifying code operations can cause unexpected instruction execution results
H50	X	X	X	NoFix	Deadlock may occur due to illegal-instruction/page-miss combination
H51	X	X	X	NoFix	FLUSH# assertion following STPCLK# may prevent CPU clocks from stopping
H52	X	X	X	NoFix	Floating-point exception condition may be deferred
H53			X	NoFix	Race conditions may exist on thermal sensor SMBus collision detection/arbitration circuitry
H54	X	X	X	NoFix	Intermittent power-on failure due to uninitialized processor internal nodes
H55	X	X	X	NoFix	Selector for the LTR/LLDT register may get corrupted
H56	X	X	X	NoFix	INIT does not clear global entries in the TLB
H57	X	X	X	NoFix	VM bit will be cleared on a double fault handler
H58	X	X	X	NoFix	Memory aliasing with inconsistent A and D bits may cause processor deadlock
H59	X	X	X	NoFix	Use of memory aliasing with inconsistent memory type may cause system hang
H60	X	X	X	NoFix	Processor may report invalid TSS fault instead of Double fault during mode C paging
H61	X	X	X	NoFix	Machine check exception may occur when interleaving code between different memory types
H62	X	X	X	NoFix	Wrong ESP Register Values During a Fault in VM86 Mode
H63	X	X	X	NoFix	APIC ICR Write May Cause Interrupt Not to be Sent When ICR Delivery Bit Pending
H64	X	X	X	NoFix	Processor Incorrectly Samples NMI Interrupt after RESET# Deassertion When Processor APIC is Hardware-Disabled

Summary of Errata

NO.	mcbA0	mcpA0	cmmA0	Plans	ERRATA
H65	X	X	X	NoFix	The Instruction Fetch Unit (IFU) May Fetch Instructions Based Upon Stale CR3 Data After a Write to CR3 Register
H66	X	X	X	NoFix	Under some complex conditions, the instructions in the Shadow of a JMP FAR may be Unintentionally Executed and Retired
H67	X	X	X	NoFix	Processor Does not Flag #GP on Non-zero Write to Certain MSRs

Summary of Documentation Changes

NO.	mcbA0	mcpA0	cmmA0	Plans	DOCUMENTATION CHANGES
H1	X	X	X	Doc	Mobile Celeron Processor CPUID Section Update
H2	X	X	X	Doc	SSE and SSE2 Instructions Opcodes
H3	X	X	X	Doc	Executing the SSE2 Variant on a Non-SSE2 Capable Processor
H4	X	X	X	Doc	Direction Flag (DF) Mistakenly Denoted as a System Flag
H5	X	X	X	Doc	Fopcode Compatibility Mode
H6	X	X	X	Doc	FCOS, FPTAN, FSIN, and FSINCOS Trigonometric Domain not Correct
H7	X	X	X	Doc	Incorrect Description of Stack
H8	X	X	X	Doc	EFLAGS Register Correction
H9	X	X	X	Doc	PSE-36 Paging Mechanism
H10	X	X	X	Doc	0x33 Opcode
H11	X	X	X	Doc	Incorrect Information for SLDT
H12	X	X	X	Doc	LGDT/LIDT Instruction Information Correction
H13	X	X	X	Doc	Errors In Instruction Set Reference
H14	X	X	X	Doc	RSM Instruction Set Summary
H15	X	X	X	Doc	Correct MOVAPS and MOVAPD Operand Section
H16	X	X	X	Doc	DAA—Decimal Adjust AL after Addition
H17	X	X	X	Doc	DAS—Decimal Adjust AL after Subtraction
H18	X	X	X	Doc	Omission of Dependency between BTM and LBR

Summary of Documentation Changes

NO.	mcbA0	mcpA0	cmmA0	Plans	DOCUMENTATION CHANGES
H19	X	X	X	Doc	I/O Permissions Bitmap Base Addy > 0xDFFF Does not Cause #GP(0) Fault
H20	X	X	X	Doc	Wrong Field Width for MINSS and MAXSS
H21	X	X	X	Doc	Figure 15-12 PEBS Record Format
H22	X	X	X	Doc	I/O Permission Bit Map
H23	X	X	X	Doc	Cache Description
H24	X	X	X	Doc	Instruction Formats and Encoding
H25	X	X	X	Doc	Machine-Check Initialization

Summary of Specification Clarifications

NO.	mcbA0	mcpA0	cmmA0	Plans	SPECIFICATION CLARIFICATIONS
H1	X	X		Doc	Shipping container maximum temperature rating for BGA1
H2			X	Doc	ESD for MMC-1 and MMC-2
H3			X	Doc	Bulk capacitance requirements for the system electronics

Summary of Specification Changes

NO.	mcbA0	mcpA0	cmmA0	Plans	SPECIFICATION CHANGES
H1	X	X		Doc	$I_{CC,SG}$, $I_{CC,QS}$, and $I_{CC,DSLP}$ maximum specifications for BGA1 and mini-cartridge
H2	X	X		Doc	Temperature note for thermal power specifications

ERRATA

H1. FP Data Operand Pointer May Be Incorrectly Calculated After FP Access Which Wraps 64-Kbyte Boundary in 16-Bit Code

Problem: The FP Data Operand Pointer is the effective address of the operand associated with the last noncontrol floating-point instruction executed by the machine. If an 80-bit floating-point access (load or store) occurs in a 16-bit mode other than protected mode (in which case the access will produce a segment limit violation), the memory access wraps a 64-Kbyte boundary, and the floating-point environment is subsequently saved, the value contained in the FP Data Operand Pointer may be incorrect.

Implication: A 32-bit operating system running 16-bit floating-point code may encounter this erratum, under the following conditions:

- The operating system is using a segment greater than 64 Kbytes in size.
- An application is running in a 16-bit mode other than protected mode.
- An 80-bit floating-point load or store which wraps the 64-Kbyte boundary is executed.
- The operating system performs a floating-point environment store (FSAVE/FNSAVE/FSTENV/FNSTENV) after the above memory access.
- The operating system uses the value contained in the FP Data Operand Pointer.

Wrapping an 80-bit floating-point load around a segment boundary in this way is not a normal programming practice. Intel has not currently identified any software which exhibits this behavior.

Workaround: If the FP Data Operand Pointer is used in an OS which may run 16-bit floating-point code, care must be taken to ensure that no 80-bit floating-point accesses are wrapped around a 64-Kbyte boundary.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H2. Differences Exist in Debug Exception Reporting

Problem: There exist some differences in the reporting of code and data breakpoint matches between that specified by previous Intel processors' specifications and the behavior of the Mobile Intel® Celeron® processor, as described below:

Case 1: The first case is for a breakpoint set on a MOVSS or POPSS instruction, when the instruction following it causes a debug register protection fault (DR7.gd is already set, enabling the fault). The processor reports delayed data breakpoint matches from the MOVSS or POPSS instructions by setting the matching DR6.bi bits, along with the debug register protection fault (DR6.bd). If additional breakpoint faults are matched during the call of the debug fault handler, the processor sets the breakpoint match bits (DR6.bi) to reflect the breakpoints matched by both the MOVSS or POPSS breakpoint and the debug fault handler call. The Mobile Intel® Celeron® processor only sets DR6.bd in either situation, and does not set any of the DR6.bi bits.

Case 2: In the second breakpoint reporting failure case, if a MOVSS or POPSS instruction with a data breakpoint is followed by a store to memory which:

a) crosses a 4-Kbyte page boundary,

OR

b) causes the page table Access or Dirty (A/D) bits to be modified,

the breakpoint information for the MOVSS or POPSS will be lost. Previous processors retain this information under these boundary conditions.

Case 3: If they occur after a MOVSS or POPSS instruction, the INT*n*, INTO, and INT3 instructions zero the DR6.bi bits (bits B0 through B3), clearing pending breakpoint information, unlike previous processors.

Case 4: If a data breakpoint and an SMI (System Management Interrupt) occur simultaneously, the SMI will be serviced via a call to the SMM handler, and the pending breakpoint will be lost.

Case 5: When an instruction which accesses a debug register is executed, and a breakpoint is encountered on the instruction, the breakpoint is reported twice.

Case 6: Unlike previous versions of Intel Architecture processors, Mobile Intel® Celeron® processors will not set the Bi bits for a matching disabled breakpoint unless at least one other breakpoint is enabled.

Implication: When debugging or when developing debuggers for a Mobile Intel® Celeron® processor-based system, this behavior should be noted. Normal usage of the MOVSS or POPSS instructions (i.e., following them with a MOV ESP) will not exhibit the behavior of cases 1-3. Debugging in conjunction with SMM will be limited by case 4.

Workaround: Following MOVSS and POPSS instructions with a MOV ESP instruction when using breakpoints will avoid the first three cases of this erratum. No workaround has been identified for cases 4, 5, or 6.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H3. Code Fetch Matching Disabled Debug Register May Cause Debug Exception

Problem: The bits L0-3 and G0-3 enable breakpoints local to a task and global to all tasks, respectively. If one of these bits is set, a breakpoint is enabled, corresponding to the addresses in the debug registers DR0 - DR3. If at least one of these breakpoints is enabled, any of these registers are *disabled* (e.g., L*n* and G*n* are 0), and RW*n* for the disabled register is 00 (indicating a breakpoint on instruction execution), normally an instruction fetch will not cause an instruction-breakpoint fault based on a match with the address in the disabled register(s). However, if the address in a disabled register matches the address of a code fetch which also results in a page fault, an instruction-breakpoint fault will occur.

Implication: While debugging software, extraneous instruction-breakpoint faults may be encountered if breakpoint registers are not cleared when they are disabled. Debug software which does not implement a code breakpoint handler will fail, if this occurs. If a handler is present, the fault will be serviced. Mixing data and code may exacerbate this problem by allowing disabled data breakpoint registers to break on an instruction fetch.

Workaround: The debug handler should clear breakpoint registers before they become disabled.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H4. Double ECC Error on Read May Result in BINIT#

Problem: For this erratum to occur, the following conditions must be met:

- Machine Check Exceptions (MCEs) must be enabled.
- A dataless transaction (such as a write invalidate) must be occurring simultaneously with a transaction which returns data (a normal read).
- The read data must contain a double-bit uncorrectable ECC error.

If these conditions are met, the Mobile Intel Celeron processor will not be able to determine which transaction was erroneous, and instead of generating an MCE, it will generate a BINIT#.

Implication: The bus will be reinitialized in this case. However, since a double-bit uncorrectable ECC error occurred on the read, the MCE handler (which is normally reached on a double-bit uncorrectable ECC error for a read) would most likely cause the same BINIT# event.

Workaround: Though the ability to drive BINIT# can be disabled in the Mobile Intel Celeron processor, which would prevent the effects of this erratum, overall system behavior would not improve, since the error which would normally cause a BINIT# would instead cause the machine to shut down. No other workaround has been identified.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H5. FP Inexact-Result Exception Flag May Not Be Set

Problem: When the result of a floating-point operation is not exactly representable in the destination format (1/3 in binary form, for example), an inexact-result (precision) exception occurs. When this occurs, the PE bit (bit 5 of the FPU status word) is normally set by the processor. Under certain rare conditions, this bit may not be set when this rounding occurs. However, other actions taken by the processor (invoking the software exception handler if the exception is unmasked) are not affected. This erratum can only occur if the floating-point operation which causes the precision exception is immediately followed by one of the following instructions:

- FST m32real
- FST m64real
- FSTP m32real
- FSTP m64real
- FSTP m80real
- FIST m16int
- FIST m32int
- FISTP m16int
- FISTP m32int
- FISTP m64int

Note that even if this combination of instructions is encountered, there is also a dependency on the internal pipelining and execution state of both instructions in the processor.

Implication: Inexact-result exceptions are commonly masked or ignored by applications, as it happens frequently, and produces a rounded result acceptable to most applications. The PE bit of the FPU status word may not always be set upon receiving an inexact-result exception. Thus, if these exceptions are unmasked, a floating-point error exception handler may not recognize that a precision exception occurred. Note that this is a “sticky” bit, e.g., once set by an inexact-result condition, it remains set until cleared by software.

Workaround: This condition can be avoided by inserting two NOP instructions between the two floating-point instructions.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H6. BTM for SMI Will Contain Incorrect FROM EIP

Problem: A system management interrupt (SMI) will produce a Branch Trace Message (BTM), if BTMs are enabled. However, the FROM EIP field of the BTM (used to determine the address of the instruction which was being executed when the SMI was serviced) will not have been updated for the SMI, so the field will report the same FROM EIP as the previous BTM.

Implication: A BTM which is issued for an SMI will not contain the correct FROM EIP, limiting the usefulness of BTMs for debugging software in conjunction with System Management Mode (SMM).

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H7. I/O Restart in SMM May Fail After Simultaneous MCE

Problem: If an I/O instruction (IN, INS, REP INS, OUT, OUTS, or REP OUTS) is being executed, and if the data for this instruction becomes corrupted, the Intel Mobile Celeron processor will signal a machine check exception (MCE). If the instruction is directed at a device which is powered down, the processor may also receive an assertion of SMI#. Since MCEs have higher priority, the processor will call the MCE handler, and the SMI# assertion will remain pending. However, upon attempting to execute the first instruction of the MCE handler, the SMI# will be recognized and the processor will attempt to execute the SMM handler. If the SMM handler is completed successfully, it will attempt to restart the I/O instruction, but will not have the correct machine state, due to the call to the MCE handler.

Implication: A simultaneous MCE and SMI# assertion may occur for one of the I/O instructions above. The SMM handler may attempt to restart such an I/O instruction, but will have corrupted state due to the MCE handler call, leading to failure of the restart and shutdown of the processor.

Workaround: If a system implementation must support both SMM and MCEs, the first thing the SMM handler code (when an I/O restart is to be performed) should do is check for a pending MCE. If there is an MCE pending, the SMM handler should immediately exit via an RSM instruction and allow the machine check exception handler to execute. If there is not, the SMM handler may proceed with its normal operation.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H8. Branch Traps Do Not Function if BTMs Are Also Enabled

Problem: If branch traps or branch trace messages (BTMs) are enabled alone, both function as expected. However, if both are enabled, only the BTMs will function, and the branch traps will be ignored.

Implication: The branch traps and branch trace message debugging features cannot be used together.

Workaround: If branch trap functionality is desired, BTMs must be disabled.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H9. Machine Check Exception Handler May Not Always Execute Successfully

Problem: An asynchronous machine check exception (MCE), such as a BINIT# event, which occurs during an access that splits a 4-Kbyte page boundary may leave some internal registers in an indeterminate state. Thus, MCE handler code may not always run successfully if an asynchronous MCE has occurred previously.

Implication: An MCE may not always result in the successful execution of the MCE handler. However, asynchronous MCEs usually occur upon detection of a catastrophic system condition that would also hang the processor. Leaving MCEs disabled will result in the condition which caused the asynchronous MCE instead causing the processor to enter shutdown. Therefore, leaving MCEs disabled may not improve overall system behavior.

Workaround: No workaround which would guarantee successful MCE handler execution under this condition has been identified.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H10. MCE Due to L2 Parity Error Gives L1 MCACOD.LL

Problem: If a Cache Reply Parity (CRP) error, Cache Address Parity (CAP) error, or Cache Synchronous Error (CSER) occurs on an access to the Mobile Intel Celeron processor's L2 cache, the resulting Machine Check Architectural Error Code (MCACOD) will be logged with '01' in the LL field. This value indicates an L1 cache error; the value should be '10', indicating an L2 cache error. Note that L2 ECC errors have the correct value of '10' logged.

Implication: An L2 cache access error, other than an ECC error, will be improperly logged as an L1 cache error in MCACOD.LL.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H11. LBER May Be Corrupted After Some Events

Problem: The last branch record (LBR) and the last branch before exception record (LBER) can be used to determine the source and destination information for previous branches or exceptions. The LBR contains the source and destination addresses for the last branch or exception, and the LBER contains similar information for the last branch taken before the last exception. This information is typically used to determine the location of a branch which leads to execution of code which causes an exception. However, after a catastrophic bus condition which results in an assertion of BINIT# and the reinitialization of the buses, the value in the LBER may be corrupted. Also, after either a CALL which results in a fault or a software interrupt, the LBER and LBR will be updated to the same value, when the LBER should not have been updated.

Implication: The LBER and LBR registers are used only for debugging purposes. When this erratum occurs, the LBER will not contain reliable address information. The value of LBER should be used with caution when debugging branching code; if the values in the LBR and LBER are the same, then the LBER value is incorrect. Also, the value in the LBER should not be relied upon after a BINIT# event.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H12. BTMs May Be Corrupted During Simultaneous L1 Cache Line Replacement

Problem: When Branch Trace Messages (BTMs) are enabled and such a message is generated, the BTM may be corrupted when issued to the bus by the L1 cache if a new line of data is brought into the L1 data cache simultaneously. Though the new line being stored in the L1 cache is stored correctly, and no corruption occurs in the data, the information in the BTM may be incorrect due to the internal collision of the data line and the BTM.

Implication: Although BTMs may not be entirely reliable due to this erratum, the conditions necessary for this boundary condition to occur have only been exhibited during focused simulation testing. Intel has currently not observed this erratum in a system level validation environment.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H13. Potential Early Deassertion of LOCK# During Split-Lock Cycles

Problem: During a split-lock cycle there are four bus transactions: 1st ADS# (a partial read), 2nd ADS# (a partial read), 3rd ADS# (a partial write), and the 4th ADS# (a partial write). Due to this erratum, LOCK# may deassert one clock after the 4th ADS# of the split-lock cycle instead of after the 4th RS# assertion corresponding to the 4th ADS# has been sampled. The following sequence of events are required for this erratum to occur:

1. A lock cycle occurs (split or nonsplit).
2. Five more bus transactions (assertion of ADS#) occur.
3. A split-lock cycle occurs and BNR# toggles after the 3rd ADS# (partial write) of the split-lock cycle. This in turn delays the assertion of the 4th ADS# of the split-lock cycle. BNR# toggling at this time could most likely happen when the bus is set for an IOQ depth of 2.

When all of these events occur, LOCK# will be deasserted in the next clock after the 4th ADS# of the split-lock cycle.

Implication: This may affect chipset logic which monitors the behavior of LOCK# deassertion.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H14. A20M# May Be Inverted After Returning From SMM and Reset

Problem: This erratum is seen when software causes the following events to occur:

1. The assertion of A20M# in real address mode.
2. After entering the 1-Mbyte address wrap-around mode caused by the assertion of A20M#, there is an assertion of SMI# intended to cause a Reset or remove power to the processor. Once in the SMM handler, software saves the SMM state save map to an area of nonvolatile memory from which it can be restored at some point in the future. Then software asserts RESET# or removes power to the processor.
3. After exiting Reset or completion of power-on, software asserts SMI# again. Once in the SMM handler, it then retrieves the old SMM state save map which was saved in event 2 above and copies it into the current SMM state save map. Software then asserts A20M# and executes the RSM instruction. After exiting the SMM handler, the polarity of A20M# is inverted.

Implication: If this erratum occurs, A20M# will behave with a polarity opposite from what is expected (e.g., the 1-Mbyte address wrap-around mode is enabled when A20M# is deasserted, and does not occur when A20M# is asserted).

Workaround: Software should save the A20M# signal state in nonvolatile memory before an assertion of RESET# or a power down condition. After coming out of Reset or at power on, SMI# should be asserted again. During the restoration of the old SMM state save map described in event 3 above, the entire map should be restored, except for bit 5 of the byte at offset 7F18h. This bit should retain the value assigned to it when the SMM state save map was created in event 3. The SMM handler should then restore the original value of the A20M# signal.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H15. Reporting of Floating-Point Exception May Be Delayed

Problem: The Mobile Intel Celeron processor normally reports a floating-point exception for an instruction when the next floating-point or MMX™ technology instruction is executed. The assertion of FERR# and/or the INT 16 interrupt corresponding to the exception may be delayed until the floating-point or MMX technology instruction *after* the one which is expected to trigger the exception, if the following conditions are met:

1. A floating-point instruction causes an exception.
2. Before another floating-point or MMX technology instruction, any one of the following occurs:
   a. A subsequent data access occurs to a page which has not been marked as accessed
   b. Data is referenced which crosses a page boundary, or
   c. A possible page-fault condition is detected which, when resolved, completes without faulting.
3. The instruction causing event 2 above is followed by a MOVQ or MOVD store instruction.

Implication: This erratum only affects software which operates with floating-point exceptions unmasked. Software which requires floating-point exceptions to be visible on the next floating-point or MMX technology instruction, and which uses floating-point calculations on data which is then used for MMX technology instructions, may see a delay in the reporting of a floating-point instruction exception in some cases. Note that mixing floating-point and MMX technology instructions in this way is not recommended.

Workaround: Inserting a WAIT or FWAIT instruction (or reading the floating-point status register) between the floating-point instruction and the MOVQ or MOVD instruction will give the expected results. This is already the recommended practice for software.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H16. Near CALL to ESP Creates Unexpected EIP Address

Problem: As documented, the CALL instruction saves procedure linking information in the procedure stack and jumps to the called procedure specified with the destination (target) operand. The target operand specifies the address of the first instruction in the called procedure. This operand can be an immediate value, a general purpose register, or a memory location. When accessing an absolute address indirectly using the stack pointer (ESP) as a base register, the base value used is the value in the ESP register before the instruction executes. However, when accessing an absolute address directly using ESP as the base register, the base value used is the value of ESP *after* the return value is pushed on the stack, not the value in the ESP register *before* the instruction executed.

Implication: Due to this erratum, the processor may transfer control to an unintended address. Results are unpredictable, depending on the particular application, and can range from no effect to the unexpected termination of the application due to an exception. Intel has observed this erratum only in a focused testing environment. Intel has not observed any commercially available operating system, application, or compiler that makes use of or generates this instruction.

Workaround: If the other seven general purpose registers are unavailable for use, and it is necessary to do a CALL via the ESP register, first push ESP onto the stack, then perform an *indirect* call using ESP (e.g., CALL [ESP]). The saved version of ESP should be popped off the stack after the call returns.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H17. Built-in Self Test Always Gives Nonzero Result

Problem: The Built-in Self Test (BIST) of the Mobile Intel Celeron processor does not give a zero result to indicate a passing test. Regardless of pass or fail status, bit 6 of the BIST result in the EAX register after running BIST is set.

Implication: Software which relies on a zero result to indicate a passing BIST will indicate BIST failure.

Workaround: Mask bit 6 of the BIST result register when analyzing BIST results.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H18. Cache State Corruption in the Presence of Page A/D-bit Setting and Snoop Traffic

Problem: If an operating system uses the Page Access and/or Dirty bit feature implemented in the Intel architecture and there is a significant amount of snoop traffic on the bus, while the processor is setting the Access and/or Dirty bit the processor may inappropriately change a single L1 cache line to the modified state.

Implication: The occurrence of this erratum may result in cache incoherency, which may cause parity errors, data corruption (with no parity error), unexpected application or operating system termination, or system hangs.

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H19. Snoop Cycle Generates Spurious Machine Check Exception

Problem: The processor may incorrectly generate a Machine Check Exception (MCE) when it processes a snoop access that does not hit the L1 data cache. Due to an internal logic error, this type of snoop cycle may still check data parity on undriven data lines. The processor generates a spurious machine check exception as a result of this unnecessary parity check.

Implication: A spurious machine check exception may result in an unexpected system halt if Machine Check Exception reporting is enabled in the operating system.

Workaround: It is possible for BIOS code to contain a workaround for this erratum. This workaround would fix the erratum; however, the data parity error will still be reported.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H20. MOVD/MOVQ Instruction Writes to Memory Prematurely

Problem: When an instruction encounters a fault, the faulting instruction should not modify any CPU or system state. However, when the MMX™ technology store instructions MOVD and MOVQ encounter any of the following events, it is possible for the store to be committed to memory even though it should be canceled:

1. If CR0.EM = 1 (Emulation bit), then the store could happen prior to the triggered invalid opcode exception.
2. If the floating-point Top-of-Stack (FP TOS) is not zero, then the store could happen prior to executing the processor assist routine that sets the FP TOS to zero.
3. If there is an unmasked floating-point exception pending, then the store could happen prior to the triggered unmasked floating-point exception.
4. If CR0.TS = 1 (Task Switched bit), then the store could happen prior to the triggered Device Not Available (DNA) exception.

If the MOVD/MOVQ instruction is restarted after handling any of the above events, then the store will be performed again, overwriting with the expected data. The instruction will not be restarted after event 1. The instruction will definitely be restarted after events 2 and 4. The instruction may or may not be restarted after event 3, depending on the specific exception handler.

Implication: This erratum causes unpredictable behavior in an application if MOVD/MOVQ instructions are used to manipulate semaphores for multiprocessor synchronization, or if these MMX instructions are used to write to uncacheable memory or memory mapped I/O that has side effects, e.g., graphics devices. This erratum is completely transparent to all applications that do not have these characteristics. When each of the above conditions are analyzed:

1. Setting the CR0.EM bit forces all floating-point/MMX instructions to be handled by software emulation. The MOVD/MOVQ instruction, which is an MMX instruction, would be considered an invalid instruction. Operating systems typically terminates the application after getting the expected invalid opcode fault.
2. The FP TOS not equal to 0 case only occurs when the MOVD/MOVQ store is the first MMX instruction in an MMX technology routine and the previous floating-point routine did not clean up the floating-point states properly when it exited. Floating-point routines commonly leave TOS to 0 prior to exiting. For a store to be executed as the first MMX instruction in an MMX technology routine following a floating-point routine, the software would be implementing instruction level intermixing of floating-point and MMX instructions. Intel does not recommend this practice.

3. The unmasked floating-point exception case only occurs if the store is the first MMX technology instruction in an MMX technology routine and the previous floating-point routine exited with an unmasked floating-point exception pending. Again, for a store to be executed as the first MMX instruction in an MMX technology routine following a floating-point routine, the software would be implementing instruction level intermixing of floating-point and MMX instructions. Intel does not recommend this practice.
4. Device Not Available (DNA) exceptions occur naturally when a task switch is made between two tasks that use either floating-point instructions and/or MMX instructions. For this erratum, in the event of the DNA exception, data from the prior task may be temporarily stored to the present task's program state.

Workaround: Do not use MMX instructions to manipulate semaphores for multiprocessor synchronization. Do not use MOVD/MOVQ instructions to write directly to I/O devices if doing so triggers user visible side effects. An OS can prevent old data from being stored to a new task's program state by cleansing the FPU explicitly after every task switch. Follow Intel's recommended programming paradigms in the *Intel Architecture Optimization Manual* for writing MMX technology programs. Specifically, do not mix floating-point and MMX instructions. When transitioning to new a MMX technology routine, begin with an instruction that does not depend on the prior state of either the MMX technology registers or the floating-point registers, such as a load or PXOR mm0, mm0. Be sure that the FP TOS is clear before using MMX instructions.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H21. Memory Type Undefined for Nonmemory Operations

Problem: The Memory Type field for nonmemory transactions such as I/O and Special Cycles are undefined. Although the Memory Type attribute for nonmemory operations logically should (and usually does) manifest itself as UC, this feature is not designed into the implementation and is therefore inconsistent.

Implication: Bus agents may decode a non-UC memory type for nonmemory bus transactions.

Workaround: Bus agents must consider transaction type to determine the validity of the Memory Type field for a transaction.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H22. FP Data Operand Pointer May Not Be Zero After Power On or Reset

Problem: The FP Data Operand Pointer, as specified, should be reset to zero upon power on or Reset by the processor. Due to this erratum, the FP Data Operand Pointer may be nonzero after power on or Reset.

Implication: Software which uses the FP Data Operand Pointer and count on its value being zero after power on or Reset without first executing an FINIT/FNINIT instruction will use an incorrect value, resulting on incorrect behavior of the software.

Workaround: Software should follow the recommendation in Section 8.2 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide* (Order Number 243192). This recommendation states that if the FPU will be used, software-initialization code should execute an FINIT/FNINIT instruction following a hardware reset. This will correctly clear the FP Data Operand Pointer to zero.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H23. MOVD Following Zeroing Instruction Can Cause Incorrect Result

Problem: An incorrect result may be calculated after the following circumstances occur:

1. A register has been zeroed with either a SUB reg, reg instruction or an XOR reg, reg instruction
2. A value is moved with sign extension into the same register's lower 16 bits; or a signed integer multiply is performed to the same register's lower 16 bits
3. This register is then copied to an MMX™ technology register using the MOVD instruction prior to any other operations on the sign-extended value.

Specifically, the sign may be incorrectly extended into bits 16-31 of the MMX technology register. Only the MMX technology register is affected by this erratum.

The erratum only occurs when the 3 following steps occur in the order shown. The erratum may occur with up to 40 intervening instructions that do not modify the sign-extended value between steps 2 and 3.

1. XOR EAX, EAX
   or SUB EAX, EAX
2. MOVSX AX, BL
   or MOVSX AX, byte ptr <memory address> or MOVSX AX, BX
   or MOVSX AX, word ptr <memory address> or IMUL BL (AX implicit, opcode F6 /5)
   or IMUL byte ptr <memory address> (AX implicit, opcode F6 /5) or IMUL AX, BX (opcode 0F AF /r)
   or IMUL AX, word ptr <memory address> (opcode 0F AF /r) or IMUL AX, BX, 16 (opcode 6B /r ib)
   or IMUL AX, word ptr <memory address>, 16 (opcode 6B /r ib) or IMUL AX, 8 (opcode 6B /r ib)
   or IMUL AX, BX, 1024 (opcode 69 /r iw)
   or IMUL AX, word ptr <memory address>, 1024 (opcode 69 /r iw) or IMUL AX, 1024 (opcode 69 /r iw)
   or CBW
3. MOVD MM0, EAX

Note that the values for immediate byte/words are merely representative (i.e., 8, 16, 1024) and that any value in the range for the size may be affected. Also, note that this erratum may occur with "EAX" replaced with any 32-bit general purpose register, and "AX" with the corresponding 16-bit version of that replacement. "BL" or "BX" can be replaced with any 8-bit or 16-bit general purpose register. The CBW and IMUL (opcode F6 /5) instructions are specific to the EAX register only.

In the example, EAX is forced to contain 0 by the XOR or SUB instructions. Since the MOVSX, IMUL and CBW instructions listed should modify only bits 15:8 of EAX by sign extension, bits 31:16 of EAX should always contain 0. This implies that when MOVD copies EAX to MM0, bits 31:16 of MM0 should also be 0. Under certain scenarios, bits 31:16 of MM0 are not 0, but are replicas of bit 15 (the 16th bit) of AX. This is noticeable when the value in AX after the MOVSX, IMUL or CBW instruction is negative, i.e., bit 15 of AX is a 1.

When AX is positive (bit 15 of AX is a 0), MOVD will always produce the correct answer. If AX is negative (bit 15 of AX is a 1), MOVD may produce the right answer or the wrong answer depending on the point in time when the MOVD instruction is executed in relation to the MOVSX, IMUL or CBW instruction.

Implication: The effect of incorrect execution will vary from unnoticeable, due to the code sequence discarding the incorrect bits, to an application failure. If the MMX technology-enabled application in which MOVD is used to manipulate pixels, it is possible for one or more pixels to exhibit the wrong color or position momentarily. It is also possible for a computational application that uses the MOVD instruction in the manner described above to produce incorrect data. Note that this data may cause an unexpected page fault or general protection fault.

Workaround: There are two possible workarounds for this erratum:

1. Rather than using the MOVSX-MOVD or CBW-MOVD pairing to handle one variable at a time, use the sign extension capabilities (PSRAW, etc.) within MMX technology for operating on multiple variables. This would result in higher performance as well.
2. Insert another operation that modifies or copies the sign-extended value between the MOVSX/IMUL/CBW instruction and the MOVD instruction as in the example below:

```
XOR EAX, EAX (or SUB EAX, EAX)
MOVSX AX, BL (or other MOVSX, other IMUL or CBW instruction)
*MOV EAX, EAX
MOVD MM0, EAX
```

*Note: MOV EAX, EAX is used here as it is fairly generic. Again, EAX can be any 32-bit register.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H24. Premature Execution of a Load Operation Prior to Exception Handler Invocation

Problem: This erratum can occur with any of the following situations:

1. If an instruction that performs a memory load causes a code segment limit violation
2. If a waiting floating-point instruction or MMX™ instruction that performs a memory load has a floating-point exception pending
3. If an MMX instruction that performs a memory load and has either CR0.EM =1 (Emulation bit set), or a floating-point Top-of-Stack (FP TOS) not equal to 0, or a DNA exception pending

If any of the above circumstances, occur it is possible that the load portion of the instruction will have executed before the exception handler is entered.

Implication: In normal code execution where the target of the load operation is to write back memory there is no impact from the load being prematurely executed, nor from the restart and subsequent re-execution of that instruction by the exception handler. If the target of the load is to uncached memory that has a system side-effect, restarting the instruction may cause unexpected system behavior due to the repetition of the side-effect.

Workaround: Code which performs loads from memory that has side-effects can effectively workaround this behavior by using simple integer-based load instructions when accessing side-effect memory and by ensuring that all code is written such that a code segment limit violation cannot occur as a part of reading from side-effect memory.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H25. Read Portion of RMW Instruction May Execute Twice

Problem: When the Mobile Intel Celeron processor executes a read-modify-write (RMW) arithmetic instruction, with memory as the destination, it is possible for a page fault to occur during the execution of the store on the memory operand after the read operation has completed but before the write operation completes.

If the memory targeted for the instruction is UC (uncached), memory will observe the occurrence of the initial load before the page fault handler and again if the instruction is restarted.

Implication: This erratum has no effect if the memory targeted for the RMW instruction has no side-effects. If, however, the load targets a memory region that has side-effects, multiple occurrences of the initial load may lead to unpredictable system behavior.

Workaround: Hardware and software developers who write device drivers for custom hardware that may have a side-effect style of design should use simple loads and simple stores to transfer data to and from the device. Then, the memory location will simply be read twice with no additional implications.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H26. Intervening Writeback May Occur During Locked Transaction

Problem: During a transaction which has the LOCK# signal asserted (e.g., a locked transaction), there is a potential for an explicit writeback caused by a previous transaction to complete while the bus is locked. The explicit writeback will only be issued by the processor which has locked the bus, and the lock signal will not be deasserted until the locked transaction completes, but the atomicity of a lock may be compromised by this erratum. Note that the explicit writeback is an expected cycle, and no memory ordering violations will occur. This erratum is, however, a violation of the bus lock protocol.

Implication: A chipset or third-party agent (TPA) which tracks bus transactions in such a way that locked transactions may only consist of a read-write or read-read-write-write locked sequence, with no transactions intervening, may lose synchronization of state due to the intervening explicit writeback. Systems using chipsets or TPAs which can accept the intervening transaction will not be affected.

Workaround: The bus tracking logic of all devices on the system bus should allow for the occurrence of an intervening transaction during a locked transaction.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H27. MC2_STATUS MSR Has Model-Specific Error Code and Machine Check Architecture Error Code Reversed

Problem: The *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide,* documents that for the MC*i*_STATUS MSR, bits 15:0 contain the MCA (machine-check architecture) error code field and bits 31:16 contain the model-specific error code field. However, for the MC2_STATUS MSR, these bits have been reversed. For the MC2_STATUS MSR, bits 15:0 contain the model-specific error code field and bits 31:16 contain the MCA error code field.

Implication: A machine check error may be decoded incorrectly if this erratum on the MC2_STATUS MSR is not taken into account.

Workaround: When decoding the MC2_STATUS MSR, reverse the two error fields.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H28. Mixed Cacheability of Lock Variables Is Problematic in MP Systems

Problem: This errata only affects multiprocessor systems where a lock variable address is marked cacheable in one processor and uncacheable in any others. The processors which have it marked uncacheable may stall indefinitely when accessing the lock variable. The stall is only encountered if:

- One processor has the lock variable cached, and is attempting to execute a cache lock.
- The processor that has that address cached has it cached in its L2 only.

Other processors, meanwhile, issue back to back accesses to that same address on the bus.

Implication: MP systems where all processors either use cache locks or consistent locks to uncacheable space will not encounter this problem. If, however, a lock variable's cacheability varies in different processors, and several processors are all attempting to perform the lock simultaneously, an indefinite stall may be experienced by the processors which have it marked uncacheable in locking the variable (if the conditions above are satisfied). Intel has only encountered this problem in focus testing with artificially generated external events. Intel has not currently identified any commercial software which exhibits this problem.

Workaround: Follow a homogenous model for the memory type range registers (MTRRs), ensuring that all processors have the same cacheability attributes for each region of memory; do not use locks whose memory type is cacheable on one processor, and uncacheable on others. Avoid page table aliasing, which may produce a nonhomogenous memory model.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H29. Thermal Sensor May Assert SMBALERT# Incorrectly

Problem: The Intel Celeron Processor Mobile Module has a thermal sensor that monitors the processor core's temperature. Please note that desktop systems could have a similar thermal device. The thermal sensor asserts SMBALERT# if the processor temperature exceeds the temperature limits set in the Alarm Threshold Registers (T_{HIGH}, T_{LOW}). It also sets the corresponding Status Register bits to identify the cause of the interrupt. Figure 1 gives one example of the how the SMBALERT# signal could be used in a system.

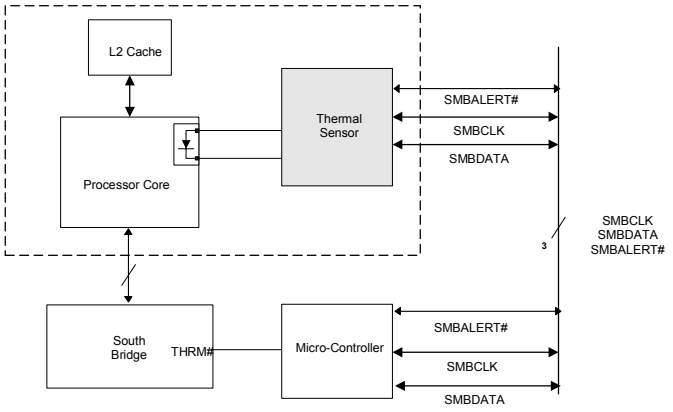


Figure 1. An Example of Microcontroller Driven Thermal Management

Implication: There is no system impact from this erratum if temperature polling is used for processor thermal management. If the SMABLERT# interrupt is employed to manage processor thermal sensing, then servicing the false interrupt may result in premature system action depending on the software and hardware implementations used. The rate of the false interrupts is less than the auto-convert rate of the thermal sensor.

Workaround: Three different (mutually exclusive) workarounds are possible:

1. Before servicing an interrupt from the thermal sensor, read and compare the processor thermal reading with the threshold limits (T_{HIGH} or T_{LOW}). Figures 2 and 3 provide basic flowcharts for the implementation of this workaround in an interrupt driven system.
2. If the firmware implemented polls the Status Register only, then before taking any action, re-read the temperature register and do a comparison with the alarm threshold limits (T_{HIGH} or T_{LOW}) to determine if the value is actually still within the temperature window.
3. Use a temperature polling scheme to monitor the processor temperature.

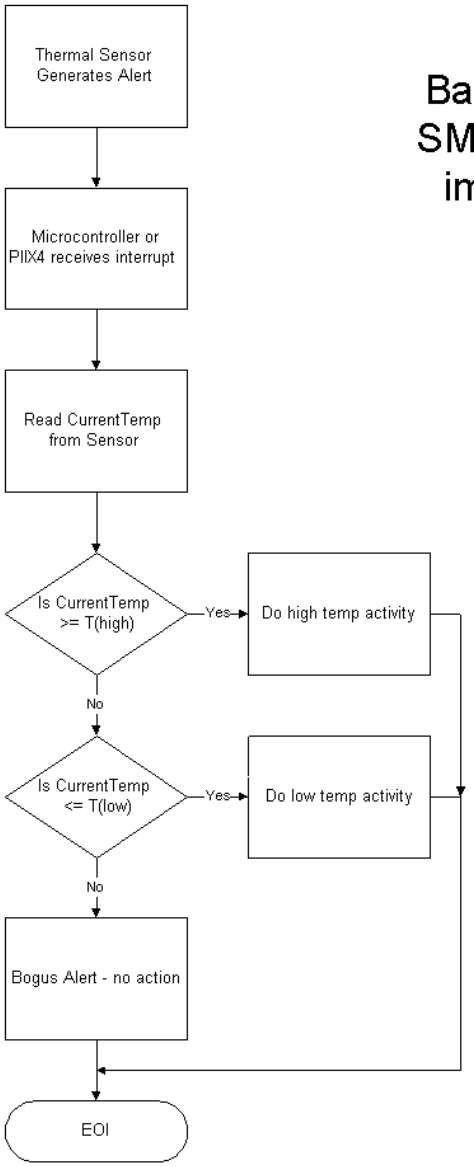


Figure 2. Workaround Flowchart: SMBALERT#-Driven System

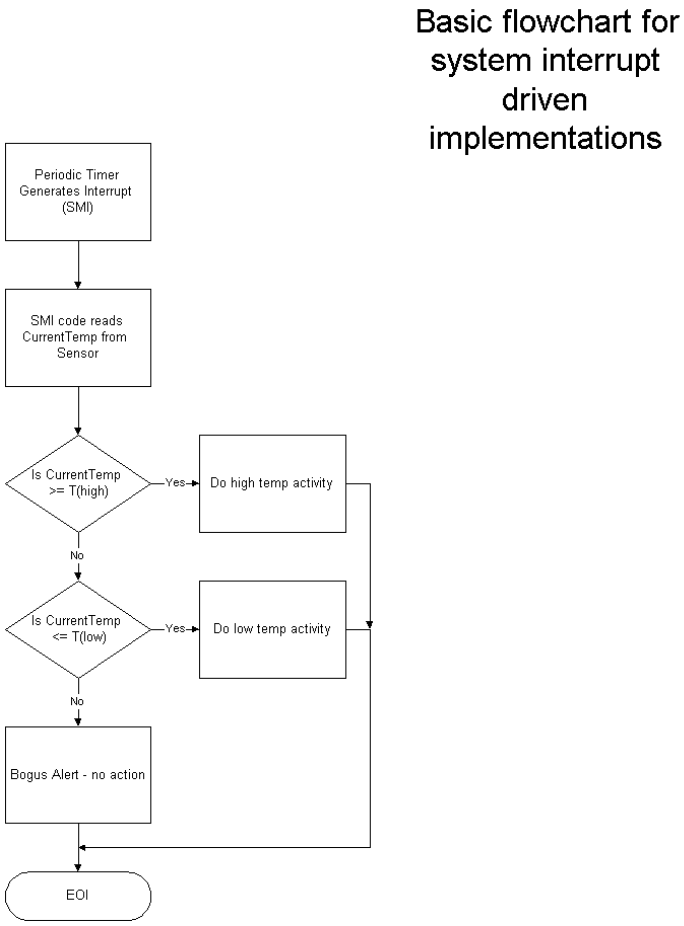


Figure 3. Workaround Flowchart: SMI#-Driven System

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H30. MOV With Debug Register Causes Debug Exception

Problem: When in V86 mode, if a MOV instruction is executed on debug registers, a general-protection exception (#GP) should be generated, as documented in the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, Section 15.2. However, in the case when the general detect enable flag (GD) bit is set, the observed behavior is that a debug exception (#DB) is generated instead.

Implication: With debug-register protection enabled (e.g., the GD bit set), when attempting to execute a MOV on debug registers in V86 mode, a debug exception will be generated instead of the expected general-protection fault.

Workaround: In general, operating systems do not set the GD bit when they are in V86 mode. The GD bit is generally set and used by debuggers. The debug exception handler should check that the exception did not occur in V86 mode before continuing. If the exception did occur in V86 mode, the exception may be directed to the general-protection exception handler.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H31. Upper Four PAT Entries Not Usable With Mode B or Mode C Paging

Problem: The Page Attribute Table (PAT) contains eight entries, which must all be initialized and considered when setting up memory types for the Mobile Intel Celeron processor. However, in Mode B or Mode C paging, the upper four entries do not function correctly for 4-Kbyte pages. Specifically, bit seven of page table entries that translate addresses to 4-Kbyte pages should be used as the upper bit of a three-bit index to determine the PAT entry that specifies the memory type for the page. When Mode B (CR4.PSE = 1) and/or Mode C (CR4.PAE) are enabled, the processor forces this bit to zero when determining the memory type regardless of the value in the page table entry. The upper four entries of the PAT function correctly for 2-Mbyte and 4-Mbyte large pages (specified by bit 12 of the page directory entry for those translations).

Implication: Only the lower four PAT entries are useful for 4KB translations when Mode B or C paging is used. In Mode A paging (4-Kbyte pages only), all eight entries may be used. All eight entries may be used for large pages in Mode B or C paging.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H32. Incorrect Memory Type May be Used When MTRRs Are Disabled

Problem: If the Memory Type Range Registers (MTRRs) are disabled without setting the CR0.CD bit to disable caching, and the Page Attribute Table (PAT) entries are left in their default setting, which includes UC- memory type (PCD = 1, PWT = 0; see the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide,* for details), data for entries set to UC- will be cached as if the memory type were writeback (WB). Also, if the page tables are set to a memory type other than UC-, then the effective memory type used will be that specified by the page tables and PAT. Any regions of memory normally forced to UC by the MTRRs (such as the VGA video region) may now be incorrectly cached and speculatively accessed.

Even if the CR0.CD bit is correctly set when the MTRRs are disabled and the PAT is left in its default state, then retries and out of order retirement of UC accesses may occur, contrary to the strong ordering expected for these transactions.

Implication: The occurrence of this erratum may result in the use of incorrect data and unpredictable processor behavior when running with the MTRRs disabled. Interaction between the mouse, cursor, and VGA video display leading to video corruption may occur as a symptom of this erratum as well.

Workaround: Ensure that when the MTRRs are disabled, the CR0.CD bit is set to disable caching. This recommendation is described in the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*. If it is necessary to disable the MTRRs, first clear the PAT register before setting the CR0.CD bit, flushing the caches, and disabling the MTRRs to ensure that UC memory type is always returned and strong ordering is maintained.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H33. Misprediction in Program Flow May Cause Unexpected Instruction Execution

Problem: To optimize performance through dynamic execution technology, the P6 architecture has the ability to predict program flow. In the event of a misprediction, the processor will normally clear the incorrect prediction, adjust the EIP to the correct location, and flush out any instructions it may have fetched from the misprediction. In circumstances where a branch misprediction occurs, the correct target of the branch has already been opportunistically fetched into the streaming buffers, and the L2 cycle caused by the evicted cache line is retried by the L2 cache, the processor may fail to flush out the retirement unit before the speculative program flow is committed to a permanent state.

Implication: The results of this erratum may range from no effect to unpredictable application or OS failure. Manifestations of this failure may result in:

- Unexpected values in EIP
- Faults or traps (e.g., page faults) on instructions that do not normally cause faults
- Faults in the middle of instructions
- Unexplained values in registers/memory at the correct EIP

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H34. Data Breakpoint Exception in a Displacement Relative Near Call May Corrupt EIP

Problem: If a misaligned data breakpoint is programmed to the same cache line as the memory location where the stack push of a near call is performed and any data breakpoints are enabled, the processor will update the stack and ESP appropriately, but may skip the code at the destination of the call. Hence, program execution will continue with the next instruction immediately following the call, instead of the target of the call.
Implication: The failure mechanism for this erratum is that the call would not be taken; therefore, instructions in the called subroutine would not be executed. As a result, any code relying on the execution of the subroutine will behave unpredictably.
Workaround: Whether enabled or not, do not program a misaligned data breakpoint to the same cache line on the stack where the push for the near call is performed.

Status: For the stepping affected see the Summary of Changes at the beginning of this section.

H35. System Bus ECC Not Functional With 2:1 Ratio

Problem: If a processor is underclocked at a core frequency to system bus frequency ratio of 2:1 and system bus ECC is enabled, the system bus ECC detection and correction will negatively affect internal timing dependencies.

Implication: If system bus ECC is enabled, and the processor is underclocked at a 2:1 ratio, the system may behave unpredictably due to these timing dependencies.

Workaround: All bus agents that support system bus ECC must disable it when a 2:1 ratio is used.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H36. Fault on REP CMPS/SCAS Operation May Cause Incorrect EIP

Problem: If either a General Protection Fault, Alignment Check Fault or Machine Check Exception occur during the first iteration of a REP CMPS or a REP SCAS instruction, an incorrect EIP may be pushed onto the stack of the event handler if all the following conditions are true:

- The event occurs on the initial load performed by the instruction(s)
- The condition of the zero flag before the repeat instruction happens to be opposite of the repeat condition (e.g., REP/REPE/REPZ CMPS/SCAS with ZF = 0 or RENE/REPNZ CMPS/SCAS with ZF = 1)
- The faulting micro-op and a particular micro-op of the REP instruction are retired in the retirement unit in a specific sequence

The EIP will point to the instruction following the REP CMPS/SCAS instead of pointing to the faulting instruction.

Implication: The result of the incorrect EIP may range from no effect to unexpected application/OS behavior.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H37. RDMSR or WRMSR To Invalid MSR Address May Not Cause GP Fault

Problem: The RDMSR and WRMSR instructions allow reading or writing of MSRs (Model Specific Registers) based on the index number placed in ECX. The processor should reject access to any reserved or unimplemented MSRs by generating #GP(0). However, there are some invalid MSR addresses for which the processor will not generate #GP(0).

Implication: For RDMSR, undefined values will be read into EDX:EAX. For WRMSR, undefined processor behavior may result.

Workaround: Do not use invalid MSR addresses with RDMSR or WRMSR.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H38. SYSENTER/SYSEXIT Instructions Can Implicitly Load "Null Segment Selector" to SS and CS Registers

Problem: According to the processor specification, attempting to load a null segment selector into the CS and SS segment registers should generate a General Protection Fault (#GP). Although loading a null segment selector to the other segment registers is allowed, the processor will generate an exception when the segment register holding a null selector is used to access memory.

However, the SYSENTER instruction can implicitly load a null value to the SS segment selector. This can occur if the value in SYSENTER_CS_MSR is between FFF8h and FFFBh when the SYSENTER instruction is executed. This behavior is part of the SYSENTER/SYSEXIT instruction definition; the content of the SYSTEM_CS_MSR is always incremented by 8 before it is loaded into the SS. This operation will set the null bit in the segment selector if a null result is generated, but it does not generate a #GP on the SYSENTER instruction itself. An exception will be generated as expected when the SS register is used to access memory, however.

The SYSEXIT instruction will also exhibit this behavior for both CS and SS when executed with the value in SYSENTER_CS_MSR between FFF0h and FFF3h, or between FFE8h and FFEBh, inclusive.

Implication: These instructions are intended for operating system use. If this erratum occurs (and the OS does not ensure that the processor never has a null segment selector in the SS or CS segment registers), the processor's behavior may become unpredictable, possibly resulting in system failure.

Workaround: Do not initialize the SYSTEM_CS_MSR with the values between FFF8h and FFFBh, FFF0h and FFF3h, or FFE8h and FFEBh before executing SYSENTER or SYSEXIT.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H39. PRELOAD Followed by EXTEST Does Not Load Boundary Scan Data

Problem: According to the IEEE 1149.1 Standard, the EXTEST instruction would use data "typically loaded onto the latched parallel outputs of boundary-scan shift-register stages using the SAMPLE/PRELOAD instruction prior to the selection of the EXTEST instruction." As a result of this erratum, this method cannot be used to load the data onto the outputs.

Implication: Using the PRELOAD instruction prior to the EXTEST instruction will not produce expected data after the completion of EXTEST.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H40. Far Jump to New TSS With D-bit Cleared May Cause System Hang

Problem: A task switch may be performed by executing a far jump through a task gate or to a new Task State Segment (TSS) directly. Normally, when such a jump to a new TSS occurs, the D-bit (which indicates that the page referenced by a Page Table Entry (PTE) has been modified) for the PTE which maps the location of the previous TSS will already be set and the processor will operate as expected. However, if the D-bit is clear at the time of the jump to the new TSS, the processor will hang.

Implication: If an OS is used which can clear the D-bit for system pages, and which jumps to a new TSS on a task switch, then a condition may occur which results in a system hang. Intel has not identified any commercial software which may encounter this condition; this erratum was discovered in a focused testing environment.

Workaround: Ensure that OS code does not clear the D-bit for system pages (including any pages that contain a task gate or TSS). Use task gates rather than jumping to a new TSS when performing a task switch.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H41. Incorrect Chunk Ordering May Prevent Execution of the Machine Check Exception Handler After BINIT#

Problem: If a catastrophic bus error is detected which results in a BINIT# assertion, and the BINIT# assertion is propagated to the processor's L2 cache at the same time that data is being sent to the processor, then the data may become corrupted in the processor's L1 cache.

Implication: Since BINIT# assertion is due to a catastrophic event on the bus, the corrupted data will not be used. However, it may prevent the processor from executing the Machine Check Exception (MCE) handler, causing the system to hang.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H42. Resume Flag May Not Be Cleared After Debug Exception

Problem: The Resume Flag (RF) is normally cleared by the processor after executing an instruction which causes a debug exception (#DB). In the process of determining whether the RF needs to be cleared after executing the instruction, the processor uses an internal register containing stale data. The stale data may unpredictably prevent the processor from clearing the RF.

Implication: If this erratum occurs, further debug exceptions will be disabled.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H43. Processor May Return Invalid Parameters on Execution of the CPUID Instruction

Problem: The Mobile Module based Intel Celeron processor with on-die L2 cache may return an incorrect "Maximum CPUID Input Value." The Mobile Module based Intel Celeron processor with on-die L2 cache is specified to return a value of 2 in the EAX register when the CPUID instruction is executed with EAX=0; however, this erratum may result in the value of 3 being returned in EAX. It is also possible that bit 18 of the EDX register will be set to a 1 when CPUID is executed with EAX=1. This bit is defined as reserved for the Mobile Module based Intel Celeron processor with on-die L2 cache, but is expected to be set to zero. If CPUID were to be executed on the Mobile Module based Intel Celeron processor with on-die L2 cache with EAX=3 the processor should return the cache parameters in the integer registers (EAX, EBX, ECX, EDX); however, the processors affected by this erratum will return undefined values in the integer registers.

Implication: Intel has not seen any software failures as a result of this erratum; however, since software written for the Mobile Module based Intel Celeron processor with on-die L2 cache will not be expecting to see a Maximum CPUID Input Value greater than 2, it is not possible to predict how software will behave on processors with this erratum. Software using the fact that bit 18 in the feature flags is set, to determine the presence of the Pentium® III processor serial number feature, without also verifying that it is executing on a Pentium III processor, may incorrectly believe that the Mobile Module based Intel Celeron processor with on-die L2 cache, support the processor serial number feature. However, the values the CPUID instruction returns when CPUID is executed with EAX=3 will not be the processor serial number and will be undefined.

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H44. Internal Cache Protocol Violation May Cause System Hang

Problem: An Mobile Intel Celeron processor based system may hang due to an internal cache protocol violation. During multiple transactions targeted at the same cacheline, there exists a small window of time such that the processor's internal timings align to create a livelock situation. The scenario, which results in the erratum, is summarized below:

Scenario:

1. A snoopable transaction is issued to address A. This snoopable transaction can be issued by the processor or the chipset.
2. The snoopable transaction hits a modified line in the processor's L1 data cache.
3. The processor issues two code fetches from the L2 cache before the snoopable transaction reaches the top of the In-Order Queue and before the snoopable transaction's modified L1 cache line containing address A is brought out on the system bus.
4. At the same time, a locked access to the L1 cache occurs.

Implication: An Mobile Intel Celeron processor may cause a system to hang if the above listed sequence of events occur. The probability of encountering this erratum increases with I/O queue depth greater than four.

Workaround: It is possible for the BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H45. GP# Fault on WSMSR to ROB_CR_BKUPTMPDR6

Problem: Writing a '1' to unimplemented bit(s) in the ROB_CR_BKUPTMPDR6 MSR (offset 1E0H) will result in a general protection fault (GP#).

Implication: The normal process used to write an MSR is to read the MSR using RDMSR, modify the bit(s) of interest, and then to write the MSR using WRMSR. Because of this erratum, this process may result in a GP# fault when used to modify the ROB_CR_BKUPTMPDR6 MSR.

Workaround: When writing to ROB_CR_BKUPTMPDR6 all unimplemented bits must be '0.' Implemented bits may be set as '0' or '1' as desired.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H46. Machine Check Exception May Occur Due to Improper Line Eviction in the IFU

Problem: The Mobile Intel Celeron processor is designed to signal an unrecoverable Machine Check Exception (MCE) as a consistency checking mechanism. Under a complex set of circumstances involving multiple speculative branches and memory accesses there exists a one cycle long window in which the processor may signal a MCE in the Instruction Fetch Unit (IFU) because instructions previously decoded have been evicted from the IFU. The one cycle long window is opened when an opportunistic fetch receives a partial hit on a previously executed but not as yet completed store resident in the store buffer. The resulting partial hit erroneously causes the eviction of a line from the IFU at a time when the processor is expecting the line to still be present. If the MCE for this particular IFU event is disabled, execution will continue normally.

Implication: While this erratum may occur on a system with any number of processors, the probability of occurrence increases with the number of processors. If this erratum does occur, a machine check exception will result. Note systems that implement an operating system that does not enable the Machine Check Architecture will be completely unaffected by this erratum, e.g., Windows* 95 and Windows 98.

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H47 Lower Bits of SMRAM SMBASE Register Cannot Be Written With an ITP

Problem: The System Management Base (SMBASE) register (7EF8H) stores the starting address of the System Management RAM (SMRAM). This register is used by the processor when it is in System Management Mode (SMM), and its contents serve as the memory base for code execution and data storage. The 32-bit SMBASE register can normally be programmed to any value. When programmed with an In-Target Probe (ITP), however, any attempt to set the lower 11 bits of SMBASE to anything other than zeros via the WRMSR instruction will cause the attempted write to fail.

Implication: When set via the ITP, any attempt to relocate SMRAM space must be made with 2 Kbyte alignment.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H48. Task Switch May Cause Wrong PTE and PDE Access Bit to be Set

Problem: If an operating system executes a task switch via a Task State Segment (TSS), and the TSS is wholly or partially located within a clean page (A and D bits clear) and the GDT entry for the new TSS is either misaligned across a cache line boundary or is in a clean page, the accessed and dirty bits for an incorrect page table/directory entry may be set.

Implication: An operating system that uses hardware task switching (or hardware task management) may encounter this erratum. The effect of the erratum depends on the alignment of the TSS and ranges from no anomalous behavior to unexpected errors.

Workaround: The operating system could align all TSSs to be within page boundaries and set the A and D bits for those pages to avoid this erratum. The operating system may alternately use software task management.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H49. Unsynchronized Cross-Modifying Code Operations Can Cause Unexpected Instruction Execution Results

Problem: The act of one processor, or system bus master, writing data into a currently executing code segment of a second processor with the intent of having the second processor execute that data as code is called cross-modifying code (XMC). XMC that does not force the second processor to execute a synchronizing instruction, prior to execution of the new code, is called unsynchronized XMC.

Software using unsynchronized XMC to modify the instruction byte stream of a processor can see unexpected instruction execution from the processor which is executing the modified code.

Implication: In this case, the phrase "unexpected execution behavior" encompasses the generation of most of the exceptions listed in the *Intel Architecture Software Developer's Manual Volume 3: System Programming Guide* including a General Protection Fault (GPF). In the event of a GPF, the application executing the unsynchronized XMC operation would be terminated by the operating system.

Workaround: In order to avoid this erratum, programmers should use the XMC synchronization algorithm as detailed in the *Intel Architecture Software Developer's Manual Volume 3: System Programming Guide*, Section 7.1.3.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H50. Deadlock May Occur Due To Illegal-Instruction/Page-Miss Combination

Problem: Intel's 32-bit Instruction Set Architecture (ISA) utilizes most of the available op-code space; however some byte combinations remain undefined and are considered illegal instructions. Intel processors detect the attempted execution of illegal instructions and signal an exception. This exception is handled by the operating system and/or application software.

Under a complex set of internal and external conditions involving illegal instructions, a deadlock may occur within the processor. The necessary conditions for the deadlock involve:

1. The illegal instruction is executed.
2. Two page table walks occur within a narrow timing window coincident with the illegal instruction.

Implication: The illegal instructions involved in this erratum are unusual and invalid byte combinations that are not useful to application software or operating systems. These combinations are not normally generated in the course of software programming, nor are such sequences known by Intel to be generated in commercially available software and tools. Development tools (compilers, assemblers) do not generate this type of code sequence, and will normally flag such a sequence as an error. If this erratum occurs, the processor deadlock condition will occur and result in a system hang. Code execution cannot continue without a system RESET.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H51. FLUSH# Assertion Following STPCLK# May Prevent CPU Clocks From Stopping

Problem: If FLUSH# is asserted after STPCLK# is asserted, the cache flush operation will not occur until after STPCLK# is de-asserted. Furthermore, the pending flush will prevent the processor from entering the Sleep state, since the flush operation must complete prior to the processor entering the Sleep state.

Implication: Following SLP# assertion, processor power dissipation may be higher than expected. Furthermore, if the source to the processor's input bus clock (BCLK) is removed, normally resulting in a transition to the Deep Sleep state, the processor may shutdown improperly. The ensuing attempt to wake up the processor will result in unpredictable behavior and may cause the system to hang.

Workaround: For systems that use the FLUSH# input signal and Deep Sleep state of the processor, ensure that FLUSH# is not asserted while STPCLK# is asserted.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H52. Floating-Point Exception Condition May Be Deferred

Problem: A floating-point instruction that causes a pending floating-point exception (ES=1) is normally signaled by the processor on the next waiting FP/MMX™ technology instruction. In the following set of circumstances, the exception may be delayed or the FSW register may contain a wrong value:

1. The excepting floating-point instruction is followed by an instruction that accesses memory across a page (4-Kbyte) boundary or its access results in the update of a page table dirty/access bit.
2. The memory accessing instruction is immediately followed by a waiting floating-point or MMX technology instruction.
3. The waiting floating-point or MMX technology instruction retires during a one-cycle window that coincides with a sequence of internal events related to instruction cache line eviction.

Implication: The floating-point exception will not be signaled until the next waiting floating-point/MMX technology instruction. Alternatively, it may be signaled with the wrong TOS and condition code values. This erratum has not been observed in any commercial software applications.

Workaround: None identified

Status: For the stepping affected see the *Summary of Changes* at the beginning of this section.

H53. Race Conditions May Exist on Thermal Sensor SMBus Collision Detection/Arbitration Circuitry

Problem: In certain SMBus configurations, when the thermal sensor is used in “hard wired alert” mode along with at least one other device on the bus, the thermal sensor may continue to send its address after losing a collision arbitration in response to an Alert Response Address (ARA) by the SMBus controller.

In order for this erratum to occur, all of the following conditions must be present:

1. The thermal sensor must be configured with alert enabled (default setting).
2. There must be one or more other devices on the SMBus along with the thermal sensor.
3. One or more of these other devices must be also configured with the alert enabled.
4. One or more of these other devices must have a lower address (higher priority) than the thermal sensor.
5. The thermal sensor must generate an SM alert while at least one other device has an SM alert pending to be serviced.

In this situation, the thermal sensor will continue to send its address on the SMBus even if it has a lower priority than the pending alert. When this occurs, the SMBus controller cannot correctly interpret the device address. This may cause the thermal sensor’s alert flag not to clear and may result in SMBus lockup.

Implication: The SMBus controller may see an invalid address and the resulting response of the SMBus controller will vary from implementation to implementation.

Workaround: Remove any one of the five conditions listed above or:

1. In software, use polling mode for the thermal sensor data collection with alert disabled. This software workaround has been validated on both Intel’s test platforms as well as on certain OEM systems.
2. Ensure that the thermal sensor alert may be cleared by a hardware or software mechanism. The implementation of this workaround will be system dependent.

Status: For the steppings affected, see the *Summary of Changes* at the beginning of this section.

H54. Intermittent Power-on Failure Due To Uninitialized Processor Internal Nodes

Problem: If there is no clock source supplied to the processor’s PICCLK pin, the processor may drive an incorrect address for the reset vector at power-on due to uninitialized processor internal nodes. In this scenario when ADS# is asserted, it is possible that the processor drives either the SMI or NMI vector addresses, rather than the reset vector address.

Implication: Systems that provide a clock to the processor’s PICCLK pin are unaffected by this issue. On a system implementation with no clock source supplied to the processor’s PICCLK pin, a small percentage of the systems may intermittently fail to boot, or may fail to resume from a STR or STD state. On the next power-on, the system will likely boot normally.

Workaround: Supply a clock source to the processor’s PICCLK pin.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H55. Selector for The LTR/LLDT Register May Get Corrupted

Problem: The internal selector portion of the respective register (TR, LDTR) may get corrupted if, during a small window of LTR or LLDT system instruction execution, the following sequence of events occur:

1. Speculative write to a segment register that might follow the LTR or LLDT instruction
2. The read segment descriptor of LTR/LLDT operation spans a page (4 Kbytes) boundary; or causes a page fault

Implication: Incorrect selector for LTR, LLDT instruction could be used after a task switch.

Workaround: Software can insert a serializing instruction between the LTR or LLDT instruction and the segment register write.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H56. INIT Does Not Clear Global Entries in The TLB

Problem: INIT may not flush a TLB entry when:

1. The processor is in protected mode with paging enabled and the page global enable flag is set (PGE bit of CR4 register)
2. G bit for the page table entry is set
3. TLB entry is present in TLB when INIT occurs

Implication: Software may encounter unexpected page fault or incorrect address translation due to a TLB entry erroneously left in TLB after INIT.

Workaround: Write to CR3, CR4 or CR0 registers before writing to memory early in BIOS code to clear all the global entries from TLB.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H57. VM Bit Will Be Cleared on a Double Fault Handler

Problem: Following a task switch to a Double Fault Handler that was initiated while the processor was in virtual-8086 (VM86) mode, the VM bit will be incorrectly cleared in EFLAGS.

Implication: When the OS recovers from the double fault handler, the processor will no longer be in VM86 mode.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H58. Memory Aliasing with Inconsistent A and D Bits May Cause Processor Deadlock

Problem: In the event that software implements memory aliasing by having two Page Directory Entries(PDEs) point to a common Page Table Entry (PTE) and the Accessed and Dirty bits for the two PDEs are allowed to become inconsistent the processor may become deadlocked.

Implication: This erratum has not been observed with commercially available software.

Workaround: Software that needs to implement memory aliasing in this way should manage the consistency of the Accessed and Dirty bits.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H59. Use of Memory Aliasing with Inconsistent Memory Type May Cause System Hang

Problem: Software that implements memory aliasing by having more than one linear addresses mapped to the same physical page with different cache types may cause the system to hang. This would occur if one of the addresses is non-cacheable used in code segment and the other a cacheable address. If the cacheable address finds its way in instruction cache, and non-cacheable address is fetched in IFU, the processor may invalidate the non-cacheable address from the fetch unit. Any micro-architectural event that causes instruction restart will expect this instruction to still be in fetch unit and lack of it will cause system hang.

Implication: This erratum has not been observed with commercially available software.

Workaround: Although it is possible to have a single physical page mapped by two different linear addresses with different memory types, Intel has strongly discouraged this practice as it may lead to undefined results. Software that needs to implement memory aliasing should manage the memory type consistency.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H60. Processor may Report Invalid TSS Fault Instead of Double Fault During Mode C Paging

Problem: When an operating system executes a task switch via a Task State Segment (TSS) the CR3 register is always updated from the new task TSS. In the mode C paging, once the CR3 is changed the processor will attempt to load the PDPTRs. If the CR3 from the target task TSS or task switch handler TSS is not valid then the new PDPTR will not be loaded. This will lead to the reporting of invalid TSS fault instead of the expected Double fault.

Implication: Operating systems that access an invalid TSS may get invalid TSS fault instead of a Double fault.

Workaround: Software needs to ensure any accessed TSS is valid.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H61. Machine Check Exception may Occur When Interleaving Code Between Different Memory Types

Problem: A small window of opportunity exists where code fetches interleaved between different memory types may cause a machine check exception. A complex set of micro-architectural boundary conditions is required to expose this window.

Implication: Interleaved instruction fetches between different memory types may result in a machine check exception. The system may hang if machine check exceptions are disabled. Intel has not observed the occurrence of this erratum while running commercially available applications or operating systems.

Workaround: Software can avoid this erratum by placing a serializing instruction between code fetches which span different memory types.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H62. Wrong ESP Register Values During a Fault in VM86 Mode

Problem: At the beginning of the IRET instruction execution in VM86 mode, the lower 16 bits of the ESP register are saved as the old stack value. When a fault occurs, these 16 bits are moved into the 32-bit ESP, effectively clearing the upper 16 bits of the ESP.

Implication: This erratum has not been observed to cause any problems with commercially available software.

Workaround: None identified.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H63. APIC ICR Write May Cause Interrupt Not to be Sent When ICR Delivery Bit Pending

Problem: If the APIC ICR (Interrupt Control Register) is written with a new interrupt command while the Delivery Status bit from a previous interrupt command is set to '1' (Send Pending), the interrupt message may not be sent out by the processor.

Implication: This erratum will cause an interrupt message not to be sent, potentially resulting in system hang.

Workaround: Software should always poll the Delivery Status bit in the APIC ICR and ensure that it is '0' (Idle) before writing a new value to the ICR.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

H64. Processor Incorrectly Samples NMI Interrupt after RESET# Deassertion When Processor APIC is Hardware-Disabled

Problem: When the processor APIC is hardware-disabled the processor may incorrectly interpret the NMI signal as an NMI interrupt, instead of a frequency strap value, starting six bus clocks after RESET# is de-asserted. This will result in a processor hang due to the NMI Handler not being installed at this time.

Implication: The system may fail to boot due to this issue.

Workaround: The processor APIC must be hardware-enabled by pulling PICD[1:0] high with separate pull up resistors and supplying PICCLK to the processor.

Status: For the steppings affected, see the *Summary of Changes* at the beginning of this section.

H65. The Instruction Fetch Unit (IFU) May Fetch Instructions Based Upon Stale CR3 Data After a Write to CR3 Register

Problem - Under a complex set of conditions, there exists a one clock window following a write to the CR3 register where-in it is possible for the iTLB fill buffer to obtain a stale page translation based on the stale CR3 data. This stale translation will persist until the next write to the CR3 register, the next page fault or execution of a certain class of instructions including RDTSC, CPUID, or IRETD with privilege level change.

Implication - The wrong page translation could be used leading to erroneous software behavior.

Workaround - Operating systems that are potentially affected can add a second write to the CR3 register.

Status - For the steppings affected, see the *Summary of Changes* at the beginning of this section.

H66. *Under some complex conditions, the instructions in the Shadow of a JMP FAR may be Unintentionally Executed and Retired*

Problem - If all of the following events happen in sequence it is possible for the system or application to hang or to execute with incorrect data.

1. The execution of an instruction, with an OPCODE that requires the processor to stall the issue of micro-instructions in the flow from the microcode sequence logic block to the instruction decode block. (a StallMS condition)

2. Exactly 63 (39 for Pre-CPUID 0x6Bx) micro-instructions later, the execution of a mispredictable branch instruction. (Jcc, LOOPcc, RET Near, CALL Near Indirect, JMP ECX=0, or JMP Near Indirect)

3. The conditional branch in event (2) is mispredicted, and furthermore the mispredicted path of execution must result in either an ITLB miss, or an Instruction Cache miss. This needs to briefly stall the issue of micro-instructions again immediately after the conditional branch until that branch prediction is corrected by the jump execution block. (a 2nd StallMS condition)

4. Along the correct path of execution, the next instruction must contain a 3rd StallMS condition at a precisely aligned point in the execution of the instruction. (CLTS, POPSS, LSS, MOV to SS)

5. A JMP FAR instruction must execute within the next 63 micro-instructions (39 Pre-CPUID 0x6BX) The intervening micro-instructions must not have any events or faults. When the instruction from event (2) retires, the StallMS condition within the event (5) instruction fails to operate correctly, and instructions in the shadow of the JMP FAR instruction could be unintentionally executed.

Implication -When this erratum occurs, system and/or application may hang or have incorrect data. In current Microsoft operating systems, this may result in a blue screen. One of four of the instructions that are required to trigger this erratum, CLTS, is a privileged instruction that is only executed by operating system or driver code. Intel has not identified any commercial software which may encounter this condition; this erratum was discovered in a focused test environment. The remaining three instructions, POPSS, LSS, and MOV to SS, are executed infrequently in modern 32-bit application code.

Workaround - None, identified at this time

Status - For the stepping affected see the Summary of Changes at the beginning of this section.

H67. Processor Does not Flag #GP on Non-zero Write to Certain MSRs

Problem - When a non-zero write occurs to the upper 32 bits of SYSENTER_EIP_MSR or SYSENTER_ESP_MSR, the processor should indicate a general protection fault by flagging #GP. Due to this erratum, the processor does not flag #GP.

Implication - The processor unexpectedly does not flag #GP on a non-zero write to the upper 32 bits of SYSENTER_EIP_MSR or SYSENTER_ESP_MSR. No known commercially available operating system has been identified to be affected by this erratum.

Workaround - None identified.

Status - For the steppings affected, see the *Summary of Changes* at the beginning of this section.

DOCUMENTATION CHANGES

The Documentation Changes listed in this section applies to the following documents:

- *P6 Family of Processors Hardware Developer's Manual*
- *Intel® Mobile Celeron® Processor in Micro-PGA and BGA Package at 466 MHz, 433 MHz, 400 MHz, 366 MHz, 333 MHz, 300 MHz and 266 MHz* datasheet
- *Intel® Celeron® Processor Mobile Module: Mobile Module Connector 1 (MMC-1) at 400 MHz, 366 MHz, 333 MHz, 300 MHz, and 266 MHz* datasheet
- *Intel® Celeron® Processor Mobile Module: Mobile Module Connector 2 (MMC-2) at 400 MHz, 366 MHz, 333 MHz, 300 MHz, and 266 MHz* datasheet
- *Intel® Mobile Celeron® Processor: Mobile Module MMC-1 at 466 MHz and 433 MHz* datasheet
- *Intel® Mobile Celeron® Processor: Mobile Module MMC-2 at 466 MHz and 433 MHz* datasheet
- *Intel Architecture Software Developer's Manual*, *Volumes 1, 2 and 3*

All Documentation Changes will be incorporated into a future version of the appropriate Intel Celeron processor documentation.

H1. Mobile Celeron Processor CPUID Section Update

The Mobile Intel® Celeron® Processor in micro-PGA and BGA Packages at 466 MHz, 433 MHz, 400 MHz, 366 MHz, 300 MHz and 266 MHz datasheet (order number 245112-005) has the following Documentation Change. The datasheet was incorrectly documented as stating the EAX register contained the values after a power-on RESET. The correct statement is after a power-on RESET, the EDX register contains the processor version information (type, family, model, stepping).

Section 2.4 Update.

The Mobile Celeron processor has the same CPUID family and model number as some Pentium II processors. The Mobile Celeron processor can be distinguished from these Pentium II processors by looking at the stepping number and the cache descriptor information. A Mobile Celeron processor has a stepping number in the range of 0AH to 0CH and an L2 cache descriptor of 041H (128-Kbyte L2 cache). If the L2 cache descriptor is 042H, then the processor is a Pentium II processor. The L2 cache must be properly initialized for the L2 cache descriptor information to be correct.

When the CPUID version information is loaded with EAX=01H, the EAX register contain the values shown in Table 2.4. After a power-on RESET, the EDX register contains the processor version information (type, family, model, stepping).

See Intel Processor Identification and the CPUID Instruction Application Note AP-485 for further information.

Table 2.4. Mobile Celeron Processor CPUID

EAX[31:0]				
Reserved[31:14]	**Type [13:12]**	**Family [11:8]**	**Model [7:4]**	**Stepping [3:0]**
X	0	6	6	A - C

After the L2 cache is initialized, the CPUID cache/TLB descriptors will be the values shown in Table 2.5.

Table 2.5. Mobile Celeron Processor CPUID Cache and TLB Descriptors

Cache and TLB Descriptors	01H, 02H, 03H, 04H, 08H, 0CH, 41H

H2. SSE and SSE2 Instructions Opcodes

The note at the end of section 2.2 in the Intel Architecture Software Developer's Manual, Vol 2: Instruction Set Reference states:

NOTE:
Some of the SSE and SSE2 instructions have three-byte opcodes. For these three-byte opcodes, the third opcode byte may be F2H, F3H, or 66H. For example, the SSE2 instruction CVTDQ2PD has the three-byte opcode F3 OF E6. The third opcode byte of these three-byte opcodes should not be thought of as a prefix, even though it has the same encoding as the operand size prefix (66H) or one of the repeat prefixes (F2H and F3H). As described above, using the operand size and repeat prefixes with SSE and SSE2 instructions is reserved.

It should state:

NOTE:
Some of the SSE and SSE2 instructions have three-byte opcodes. For these three-byte opcodes, the third opcode byte may be F2H, F3H, or 66H. For example, the SSE2 instruction CVTDQ2PD has the three-byte opcode F3 OF E6. The third opcode byte of these three-byte opcodes should not be thought of as a prefix, even though it has the same encoding as the operand size prefix (66H) or one of the repeat prefixes (F2H and F3H). As described above, using the operand size and repeat prefixes with SSE and SSE2 instructions is reserved. It should also be noted that execution of SSE2 instructions on an Intel processor that does not support SSE2 (CPUID Feature flag register EDX bit 26 is clear) will result in unpredictable code execution.

H3. Executing the SSE2 Variant on a Non-SSE2 Capable Processor

In Intel Architecture Software Developer's Manual, Vol 2: Instruction Set Reference the section for each of the following instructions states that executing the instruction in real or protected mode on a processor for which the SSE2 feature flag returned by CPUID is 0 (SSE2 not supported by the processor) will result in the generation of an undefined opcode exception (#UD). This is incorrect. The SSE2 form of these instructions is defined by opcodes for which the leading opcode byte maps into an operand size prefix. Executing the SSE2 variant of these instructions on a non-SSE2 capable processor will result in an SSE like operation and not a #UD. Refer to section 2.2 of the Intel Architecture Software Developer's Manual, Vol 2 for more detail.
Instructions:

MOVD xmm, r32; MOVD r32, xmm; MOVDQA; MOVDQU; MOVQ xmm, m64; PACKSSWB/DW; PACKUSWB; PADDB/W/D; PADDSB/W; PADDUSB/W; PAND; PANDN; PCMPEQB/W/D; PCMPGTB/W/D; PMADDWD; PMULHW; PMULLW; POR; PSLLW/D/Q; PSRAW/D; PSRLW/D/Q; PSUBB/W/D; PSUBSB/W; PSUBUSB/W; PUNPCKHBW/WD/DQ; PUNPCKLBW/WD/DQ; PXOR.

H4. Direction Flag (DF) Mistakenly Denoted as a System Flag

The *Intel Architecture Software Developer's Manual, Vol 1: Basic Architecture* Section 3.4.3 "EFLAGS Register", in Figure 3-7 EFLAGS Register currently states:

X Direction Flag(DF)

It should state:

C Direction Flag(DF)

H5. Fopcode Compatibility Mode

The *Intel Architecture Software Developer's Manual, Vol 1: Basic Architecture* Section 8.1.8.1 "FOPCODE COMPATIBILITY MODE" currently states:

"When the FOP code compatibility mode is enabled, the IA32 architecture guarantees that if an unmasked x87 FPU floating-point exception is generated, the opcode of the last non-control instruction executed prior to the generation of the exception will be stored in the x87 FPU opcode register, and that value can be read by a subsequent FSAVE of FXSAVE instruction. When the fop compatibility mode is disabled (default), the value stored in the x87 FPU opcode register is undefined (reserved)."

It should state:

"If FOP code compatibility mode is enabled, the FOP is defined as it has always been in previous IA32 implementations (always defined as the FOP of the last non-transparent FP instruction executed before a FSAVE/FSTENV/FXSAVE).

If FOP code compatibility mode is disabled (default), FOP is only valid if the last non-transparent FP instruction executed before a FSAVE/FSTENV/FXSAVE had an unmasked exception."

H6. FCOS, FPTAN, FSIN, and FSINCOS Trigonometric Domain not Correct

The *Intel Architecture Software Developer's Manual, Vol 2: Instruction Set Reference* Section 3.2 "INSTRUCTION REFERENCE" FCOS, FPTAN, FSIN, and FSINCOS trigonometric domain for C2 is incorrect. Under the FPU Flags affected, C2 currently states:

C2 Set to 1 if source operand is outside the range -2^{63} to $+2^{63}$; otherwise, cleared to 0.

It should state:

C2 Set to 1 if outside range -2^{63} <source operand $<+2^{63}$; otherwise, set to 0.

H7. Incorrect Description of Stack

The *Intel Architecture Software Developer's Manual, Volume 1: Basic Architecture* Chapter 6, Section 6.2 paragraph 2, labeled "STACK" currently states:

The next available memory location on the stack is called the top of stack. At any given time, the stack pointer (contained in the ESP register) gives the address (that is the offset from the base of the SS segment) of the top of the stack.

This paragraph is incorrect and will be removed from the section listed above.

H8. EFLAGS Register Correction

The *Intel Architecture Software Developer's Manual, Volume 1: Basic Architecture* Section 3.7.2, Figure 3.7 "EFLAGS Register", currently states:

Bit 11 "OF" as "X"

It should state:

Bit 11 "OF" as "S"

H9. PSE-36 Paging Mechanism

The *Intel Architecture Software Developer's Manual, Vol 3: System Programming Guide* Chapter 3, Section 3.9, third paragraph currently states:

As is shown in Table 3-3, the following flags must be set or cleared to enable the PSE-36 paging mechanism:

- PSE-36 CPUID feature flag-When set, it indicates the availability of the PSE-36 paging mechanism on the IA-32 processor on which the CPUID instruction is executed.

- PG flag (bit 31) in register CR0-Set to 1 to enable paging.
- PSE flag (bit 4) in control register CR4 - Set to 1 to enable the page size extension for 4-Mbyte pages.
- PAE flag (bit 5) in control register CR4-Clear to 0 to disable the PAE paging mechanism.

It should state:

As is shown in Table 3-3, the following flags must be set or cleared to enable the PSE-36 paging mechanism:

- PSE-36 CPUID feature flag-When set, it indicates the availability of the PSE-36 paging mechanism on the IA-32 processor on which the CPUID instruction is executed.
- PG flag (bit 31) in register CR0-Set to 1 to enable paging.
- PAE flag (bit 5) in control register CR4-Clear to 0 to disable the PAE paging mechanism.
- PSE flag (bit 4) in control register CR4 and the PS flag in PDE- Set to 1 to enable the page size extension for 4-Mbyte pages.
- Or the PSE flag (bit 4) in control register CR4- Set to 1 and the PS flag (bit 7) in PDE- Set to 0 to enable 4-KByte pages with 32-bit addressing (below 4GBytes).

H10. OX33 Opcode

The *Intel Architecture Software Developer's Manual, Vol 2: Instruction Set Reference* Appendix A, Table A-2, the opcode corresponding to 0x33 currently states:
Gb, Ev
It should state:
Gv, Ev

Also, Page 3-791, XOR-Logical Exclusive OR, the two entries for opcode 33 currently states:

Opcode	**Instruction**	**Description**
33 /r	XOR r16,r/m16	r8 XOR r/m8
33 /r	XOR r32,r/m32	r8 XOR r/m8

It should state:

Opcode	**Instruction**	**Description**
33 /r	r16 XOR r/m16	r8 XOR r/m8
33 /r	r32 XOR r/m32	r8 XOR r/m8

H11. Incorrect Information for SLDT

In the *Intel Architecture Software Developer's Manual, Vol 2: Instruction Set Reference,* the Opcode/Instruction/Description table for SLDT currently states “SLDT *r/m32* Store segment selector from LDTR in low-order 16 bits of *r/m32*” but should instead state “SLDT *r32* Store segment selector from LDTR in low-order 16 bits of *r32*.”

H12. LGDT/LIDT Instruction Information Correction

In the *Intel Architecture Software Developer's Manual, Vol 2: Instruction Set Reference,* the sentence in the LGDT/LIDT instruction section currently states:

”See 'SFENCE -- Store Fence' in this chapter for information on storing the contents of the GDTR and IDTR."

It should state:
"See 'SGDT/SIDT' in this chapter for information on storing the contents of the GDTR and IDTR."

H13. Errors in Instruction Set Reference

The following changes will be made to the *Intel Architecture Software Developer's Manual, Vol 2: Instruction Set Reference:*

1. *Page 3-586 “PMULUDQ—Multiply Packed Unsigned Doubleword Integers”*
   currently states:

   66 OF F4 /r PMULUDQ xmm1, xmm2/m128

   It should state:

   66 0F F4 /r PMULUDQ xmm1, xmm2/m128

2. *Page A-9, Table A-3, Two-byte Opcode Map:08H-7FH (First Byte is 0FH), e*ntry 2B currently states:

   MOVNTPS
   Wps, Vps
   MOVNTPS (66)
   Wpd, Vpd

   It should state:

   MOVNTPS
   Wps, Vps
   MOVNTPD (66)
   Wpd, Vpd

3. *Page A-9, Table A-3, Two-byte Opcode Map:08H-7FH (First Byte is 0FH).*
   Entry 3C currently states:

   Blank (empty space)

   It should state:

   MOVNTI

4. *Page A-10, Table A-3, Two-byte Opcode Map:80H-7FH (First Byte is 0FH).*
   Entry D7 currently states:

   PMOVMSKB
   Gd, Pq
   PMOVMKSB (66)
   Gd, Vdq

   It should state:

   PMOVMSKB
   Gd, Pq
   PMOVMSKB (66)
   Gd, Vdq

5. *Page A-10, Table A-3, Two-byte Opcode Map:80H-7FH (First Byte is 0FH).*
   Entry F7 currently states:

   MASKMOVQ
   Ppi, Qpi
   MASKMOVQU (66)
   Vdq, Wdq

   It should state:

   MASKMOVQ
   Ppi, Qpi
   MASKMOVDQU (66)
   Vdq, Wdq

6. *Page A-11, Table A-3, Two-byte Opcode Map:88H-7FH (First Byte is 0FH).*
   The title table currently states:

   Table A-3. Two-byte Opcode Map:88H-7FH (First Byte is FFH)

   It should state:

   Table A-3. Two-byte Opcode Map:88H-7FH (First Byte is 0FH)

7. *Page A-11, Table A-3, Two-byte Opcode Map:88H-7FH (First Byte is 0FH).*
   Entry FB currently states:

   PSUBD
   Pq, Qq
   PSUBD (66)
   Vdq, Wdq

   It should state:

   PSUBQ
   Pq, Qq
   PSUBQ (66)
   Vdq, Wdq

8. *Page B-21, Table B-12, MMX Instruction Formats and Encodings (Contd.).*
   Entry PMADD currently states:

   PMADD – Packed Multiply add

   It should state:

   PMADDWD – Packed Multiply add

9. *Page B-21, Table B-12, MMX Instruction Formats and Encodings (Contd.).*
   Entry PMULH currently states:

   PMULH – Packed multiplication

   It should state:

   PMULHW – Packed multiplication, store high word

10. *Page B-21, Table B-12, MMX Instruction Formats and Encodings (Contd.).*
    Add instruction PMULHUW :

    PMULHUW – Packed multiplication, store high word (unsigned)

mmxreg2 to mmxreg1	0000 1111: 1110 0100: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111: 1110 0100: mod mmxreg r/m

11. *Page B-21, Table B-12, MMX Instruction Formats and Encodings (Contd.).*
    Entry PMULL currently states:

    PMULL – Packed multiplication

    It should state:

    PMULLW – Packed multiplication, store low word

12. *Page B-40, Table B-19, Formats and Encodings of the SSE2 SIMD Integer Instruction.*
    Entry PMADD currently states:

    PMADD – Packed multiply add

    It should state:

    PMADDWD – Packed multiply add

13. *Page B-41, Table B-19, Formats and Encodings of the SSE2 SIMD Integer Instruction.*
    Entry PMULH currently states:

    PMULH – Packed multiplication

It should state:

PMULHW – Packed multiplication, store high word

14. *Page B-41, Table B-19, Formats and Encodings of the SSE2 SIMD Integer Instruction.*
Add instruction PMULHUW:

PMULHUW – Packed multiplication, store high word (unsigned)

xmmreg2 to xmmreg1	0110 0110 : 0000 1111 : 11110 0100 : 11 xmmreg1 xmmreg2
memory to xmmreg	0110 0110 : 0000 1111 : 1110 0100 : mod xmmreg r/m

15. *Page B-41, Table B-19, Formats and Encodings of the SSE2 SIMD Integer Instruction.*
Entry PMULL currently states:

PMULL – Packed multiplication

It should state:

PMULLW – Packed multiplication, store low word

H14. RSM Instruction Set Summary

The *Intel Architecture Software Developer's Manual, Vol 1: Basic Architecture* Section 5.8 "INSTRUCTION SET SUMMARY" currently states:

RSM Return from system management mode (SSM)
It should state:

RSM Return from system management mode (SMM)

H15. Correct MOVAPS and MOVAPD Operand Section

The *Intel Architecture Software Developer's Manual, Vol 2: Instruction Set Reference* Section 3.2 "INSTRUCTION REFERENCE" MOVAPS and MOVAPD operation section currently states:
Operation
DEST ← SRC;
It should state:
Operation
DEST ← SRC;
* #GP if SRC or DEST unaligned memory operand *;

H16. DAA—Decimal Adjust AL after Addition

The *Intel Architecture Software Developer's Manual, Vol 2: Instruction Set Reference*, page 3-173 currently states:

Operation

```
IF (((AL AND 0FH) > 9) or AF = 1)
  THEN
    AL ←AL + 6;
    CF ←CF OR CarryFromLastAddition; (* CF OR carry from AL ←AL + 6 *)
    AF ←1;
  ELSE
    AF ←0;
FI;
IF ((AL AND F0H) > 90H) or CF = 1)
  THEN
    AL ←AL + 60H;
    CF ←1;
  ELSE
    CF ←0;
FI;
```

It should state:

Operation

```
old_AL ← AL;
old_CF ← CF;
CF ← 0;
IF (((AL AND 0FH) > 9) or AF = 1)
  THEN
    AL ←AL + 6;
    CF ←old_CF or (Carry from AL ←AL + 6);
    AF ←1;
  ELSE
    AF ←0;
FI;
IF ((old_AL > 99H) or (old_CF = 1)
  THEN
    AL ←AL + 60H;
    CF ←1;
  ELSE
    CF ←0;
FI;
```

H17. DAS—Decimal Adjust AL after Subtraction

The *Intel Architecture Software Developer's Manual, Vol 2: Instruction Set Reference*, page 3-175 currently states:

Operation

```
IF (AL AND 0FH) > 9 OR AF = 1
  THEN
    AL ←AL - 6;
    CF ←CF OR CarryFromLastAddition; (* CF OR carry from AL ←AL - 6 *)
    AF ←1;
  ELSE AF ←0;
FI;
IF ((AL > 9FH) or CF = 1)
  THEN
    AL ←AL - 60H;
    CF ←1;
  ELSE CF ←0;
FI;
```

It should state:

Operation

```
old_AL ← AL;
old_CF ← CF;
CF ← 0;
IF (((AL AND 0FH) > 9) or AF = 1)
  THEN
    AL ←AL - 6;
    CF ←old_CF or (Borrow from AL ←AL - 6);
    AF ←1;
  ELSE
    AF ←0;
FI;
IF ((old_AL > 99H) OR (old_CF = 1))
  THEN
    AL ←AL - 60H;
    CF ←1;
  ELSE
    CF ←0;
FI;
```

H18. Omission of Dependency between BTM and LBR

The *Intel Architecture Software Developer's Manual, Vol 3: System Programming Guide* Chapter 15, Section 5.3, page 15-15 currently states:

15.5.3. Monitoring Branches, Exceptions, and Interrupts (Pentium 4 and Intel Xeon Processors)
When the LBR flag in the IA32_DEBUGCTL MSR is set, the processor automatically begins recording branch records for taken branches, interrupts, and exceptions (except for debug exceptions) in the LBR stack MSRs.

When the processor generates a debug exception (#DB), it automatically clears the LBR flag before executing the exception handler, but does not touch the LBR stack MSRs. The branch records for the last four taken branches, interrupts, and/or exceptions are thus retained for anal-ysis by the debugger program.
The debugger can use the linear addresses in the LBR stack to reset breakpoints in the break-point-address registers (DR0 through DR3), allowing a backward trace from the manifestation of a particular bug toward its source.
Before resuming program execution from a debug-exception handler, the handler must set the LBR flag again to re-enable last branch recording.
It should state:

15.5.3. Monitoring Branches, Exceptions, and Interrupts (Pentium 4 and Intel Xeon Processors)
When the LBR flag in the IA32_DEBUGCTL MSR is set, the processor automatically begins recording branch records for taken branches, interrupts, and exceptions (except for debug exceptions) in the LBR stack MSRs.

When the processsor generates a debug exception (#DB), it automatically clears the LBR flag before executing the exception handler. This action does not clear previously stored LBR stack MSRs. The branch record for the last four taken branches, interrupts and/or exceptions are retained for analysis.
A debugger can use the linear addresses in the LBR stack to reset breakpoints in the break-point address registers (DR0 through DR3). This allows a backward trace from the manifestation of a particular bug toward its source.

If the LBR flag is cleared and TR flag in the IA32_DEBUGCTLTR MSR remains set, the processor will continue to update LBR stack MSRs. This is because BTM information must be generated from entries in the LBR stack (see 14.5.5). A #DB does not automatically clear the TR flag.

H19. I/O Permissions Bitmap Base Addy > 0xDFFF Does not Cause #GP(0) Fault

The *Intel Architecture Software Developer's Manual, Vol 1: Basic Architecture*, page 12-6, section 12.5.2, last paragraph currently states:

If the I/O bit map base address is greater than or equal to the TSS segment limit, there is no I/O permission map, and all I/O instructions generate exceptions when the CPL is greater than the current IOPL. The I/O bit map base address must be less than or equal to DFFFH.

It should state:

If the I/O bit map base address is greater than or equal to the TSS segment limit, there is no I/O permission map, and all I/O instructions generate exceptions when the CPL is greater than the current IOPL.

H20. Wrong Field Width for MINSS and MAXSS

The *Intel Architecture Software Developer's Manual, Vol 2:Instruction Set Reference* Section 3.2 Instruction Reference under "MAXSS—Return Maximum Scalar Single-Precision Floating-Point Value" page 3-415 currently states:

DEST[63-0] .IF ((DEST[31-0] = 0.0) AND (SRC[31-0] = 0.0)) THEN SRC[31-0]

It should state:

DEST[31-0] .IF ((DEST[31-0] = 0.0) AND (SRC[31-0] = 0.0)) THEN SRC[31-0]

The *Intel Architecture Software Developer's Manual, Vol 2 Section 3.2 Instruction Reference* under title "MINSS—Return Minimum Scalar Single-Precision Floating-Point Value" page 3-428 currently states:

DEST[63-0] .IF ((DEST[31-0] = 0.0) AND (SRC[31-0] = 0.0)) THEN SRC[31-0]

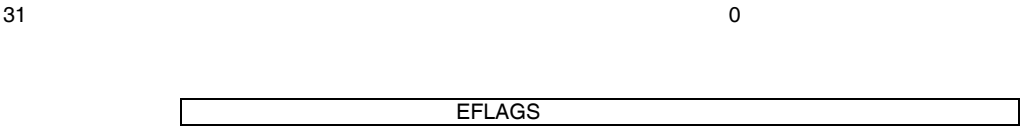
It should state:

DEST[31-0] .IF ((DEST[31-0] = 0.0) AND (SRC[31-0] = 0.0)) THEN SRC[31-0]

H21. Figure 15-12 PEBS Record Format

The *Intel Architecture Software Developer's Manual, Vol 3*: *System Programming Guide* Section 15.9.6 *" Programming the Performance Counters for Non-Retirement Events"* page 15-37, Figure 15-12, first row currently states:

It should state:



H22. I/O Permission Bit Map

The *Intel Architecture Software Developer's Manual, Vol 1: Basic Architecture,* Chapter 12, section 12.5.2, Figure 12-2 (I/O Permission Bit Map) currently states:

Last byte of bit map must be followed by a byte with all bits.
It should state:
Last byte of bit map must be followed by a byte with all bits set.

Also, in the lower left hand corner of Figure 12-2 (I/O Permission Bit Map) currently states*:*

Last I/O base map must be

It should state:

Last I/O base map must be less than or equal to DFFFH

H23. Cache Description

The *Intel Architecture Software Developer's Manual, Vol 2: Instruction Set Reference*, Table 3-10, the "sectored, 64 byte line size" description is used for the following descriptors: 0x22, 0x23, 0x79, 0x7a, 0x7b, 0x7c. This description will change to "dual-sectored line, 64 byte sector size" for clarity.

H24. Instruction Formats and Encoding

The *Intel Architecture Software Developer's Manual, Vol 2: Instruction Set Reference*, Page B-8, CMOVcc memory to register should be encoded as "0000 1111 : 0100 tttn : mod reg r/m". Page B-8, CMP immediate with memory should be encoded as "1000 00sw : mod 111 r/m : immediate data". Page B-12 POP "segment register CS, DS, ES" should be encoded as "segment register DS, ES".

H25. Machine-Check Initialization

The *Intel Architecture Software Developer's Manual, Vol 3: System Programming Guide* section 14.5 currently states*:*

14.5 MACHINE-CHECK INITIALIZATION
To use the processors machine-check architecture, software must initialize the processor to activate the machine-check exception and the error-reporting mechanism.

Example gives pseudocode for performing this initialization. This pseudocode checks for the existence of the machine-check architecture and exception on the processor, then enables the machine-check exception and the error-reporting register banks. The pseudocode assumes that the machine-check exception (#MC) handler has been installed on the system. This initialization procedure is compatible with the Pentium 4, Intel Xeon, P6 family, and Pentium processors.
Following power up or power cycling, the IA32_MC*i*_STATUS registers are not guaranteed to have valid data until after the registers are initially cleared to all 0s by software, as shown in the initialization pseudocode in Example .

Machine-Check Initialization Pseudocode

```
EXECUTE the CPUID instruction;
READ bits 7 (MCE) and 14 (MCA) of the EDX register;
IF CPU supports MCE
    THEN
        IF CPU supports MCA
            THEN
                IF IA32_ MCG_CAP.MCG_CTL_P = 1
                (* IA32_MCG_CTL register is present *)
                    IA32_MCG_CTL  FFFFFFFFFFFFFFFFH;
                    (* enables all MCA features *)
                FI;
                COUNT  IA32_MCG_CAP.Count;
                MAX_BANK_NUMBER  COUNT  1;
                (* determine number of error-reporting banks supported *)
                IF (P6 Family Processor)
                    THEN
                        FOR error-reporting banks (1 through MAX_BANK_NUMBER) DO
                            IA32_MCi_CTL  FFFFFFFFFFFFFFFFH;
                            (* enables logging of all errors except for MC0_CTL register *)
                        OD
                    ELSE (* Pentium 4 and Intel Xeon Processors *)
                        FOR error-reporting banks (0 through MAX_BANK_NUMBER) DO
                            IA32_MCi_CTL  FFFFFFFFFFFFFFFFH;
                            (* enables logging of all errors including MC0_CTL register *)
                        OD
                FI;
                FOR error-reporting banks (0 through MAX_BANK_NUMBER) DO
                    IA32_MCi_STATUS  0000000000000000H; (* clears all errors *)
                OD
        FI;
    Set the MCE flag (bit 6) in CR4 register to enable machine-check exceptions;
FI;
```

It should state:

14.5 MACHINE-CHECK INITIALIZATION

To use the processors machine-check architecture, software must initialize the processor to activate the machine-check exception and the error-reporting mechanism.
Example gives pseudocode for performing this initialization. This pseudocode checks for the existence of the machine-check architecture and exception on the processor, then enables the machine-check exception and the error-reporting register banks. The pseudocode shown is compatible with the Pentium 4, Intel Xeon, P6 family, and Pentium processors.

Following power up or power cycling, the IA32_MC*i*_STATUS registers are not guaranteed to have valid data until after the registers are initially cleared to all 0s by software, as shown in the initialization pseudocode in Example . In addition, when using P6 family processors, the software must set MCi_STATUS registers to 0 when doing a soft-reset.

Machine-Check Initialization Pseudocode

```
Check CPUID Feature Flags for MCE and MCA support
IF CPU supports MCE
THEN
    IF CPU supports MCA
    THEN
        IF (IA32_MCG_CAP.MCG_CTL_P = 1)
        (* IA32_MCG_CTL register is present *)
        THEN
            IA32_MCG_CTL  FFFFFFFFFFFFFFFFH;
            (* enables all MCA features *)
        FI

        (* Determine number of error-reporting banks supported *)
        COUNT IA32_MCG_CAP.Count;
        MAX_BANK_NUMBER  COUNT - 1;

        IF (Processor Family is 6H)
        THEN
            (* Enable logging of all errors except for MC0_CTL register *)
            FOR error-reporting banks (1 through MAX_BANK_NUMBER)
            DO
                IA32_MCi_CTL  0FFFFFFFFFFFFFFFFH;
            OD

            (* Clear all errors *)
            FOR error-reporting banks (0 through MAX_BANK_NUMBER)
            DO
                IA32_MCi_STATUS  0;
            OD

        ELSE IF (Processor Family is 0FH) (*any Processor Extended Family *)
        THEN
            (* Enable logging of all errors including MC0_CTL register *)
            FOR error-reporting banks (0 through MAX_BANK_NUMBER)
            DO
                IA32_MCi_CTL  0FFFFFFFFFFFFFFFFH;
            OD

            (* BIOS clears all errors only on power-on reset *)
            IF (BIOS detects Power-on reset)
            THEN
                FOR error-reporting banks (0 through MAX_BANK_NUMBER)
                DO
                    IA32_MCi_STATUS  0;
                OD
            ELSE
                FOR error-reporting banks (0 through MAX_BANK_NUMBER)
                DO
                    (Optional for BIOS and OS) Log valid errors
```

```
                    (OS only) IA32_MCi_STATUS  0;
                OD

            FI
        FI
    FI

    Setup the Machine Check Exception (#MC) handler for vector 18 in IDT

    Set the MCE bit (bit 6) in CR4 register to enable Machine-Check Exceptions
FI
```

SPECIFICATION CLARIFICATIONS

The Specification Clarifications listed in this section apply to the following documents:

- *P6 Family of Processors Hardware Developer's Manual*
- *Intel® Mobile Celeron® Processor in Micro-PGA and BGA Package at 466 MHz, 433 MHz, 400 MHz, 366 MHz, 333 MHz, 300 MHz and 266 MHz* datasheet
- *Intel® Celeron® Processor Mobile Module: Mobile Module Connector 1 (MMC-1) at 400 MHz, 366 MHz, 333 MHz, 300 MHz, and 266 MHz* datasheet
- *Intel® Celeron® Processor Mobile Module: Mobile Module Connector 2 (MMC-2) at 400 MHz, 366 MHz, 333 MHz, 300 MHz, and 266 MHz* datasheet
- *Intel® Mobile Celeron® Processor: Mobile Module MMC-1 at 466 MHz and 433 MHz* datasheet
- *Intel® Mobile Celeron® Processor: Mobile Module MMC-2 at 466 MHz and 433 MHz* datasheet
- *Intel Architecture Software Developer's Manual*, *Volumes 1, 2 and 3*

All Specification Clarifications will be incorporated into a future version of the appropriate Intel Celeron processor documentation.

H1. Shipping Container Maximum Temperature Rating for BGA1

In the *Intel® Mobile Celeron® Processor in BGA Package at 300 MHz and 266 MHz* datasheet, note 1 is added to Table 3.5 of the datasheets to include the maximum shipping container temperature rating of 65°C. The following is the updated Table 3.5.

Table 3.5. Mobile Celeron® Processor Absolute Maximum Ratings

Symbol	Parameter	Min	Max	Unit	Notes
$T_{Storage}$	Storage Temperature	–40	85	°C	1
V_{CC}(Abs)	Supply Voltage with respect to V_{SS}	–0.5	3.0	V	
V_{CCP}	CMOS Reference Voltage with respect to V_{SS}	–0.3	3.0	V	
V_{IN}	GTL+ Buffer DC Input Voltage with respect to V_{SS}	–0.3	V_{CC} + 0.7	V	2
V_{IN25}	2.5 V Buffer DC Input Voltage with respect to V_{SS}	–0.3	3.3	V	3

NOTES:

1. The shipping container is only rated for 65° C.
2. Parameter applies to the Low Power GTL+ signal groups only.
3. Parameter applies to CMOS, Open-Drain, APIC and TAP bus signal groups only.

H2. ESD for MMC-1 and MMC-2

In the *Intel® Mobile Celeron® Processor: Mobile Module MMC-1* datasheet and the *Intel® Mobile Celeron® Processor: Mobile Module MMC-2* datasheet, ESD is a nonpowered test of the module for noncatastrophic failure only. The module is tested at 2 KV and then is inserted in a system for a functional test.

H3. Bulk Capacitance Requirements For The System Electronics

This is a clarification to *the Intel® Celeron Processor: Mobile Module MMC-1 (Order Number #245101), the Intel® Mobile Celeron Processor: Mobile Module MMC-2 (Order Number #245102):*

> Placement of sufficient bulk capacitance on the system electronics board is critical to the operation of the Intel Celeron processor mobile module and to ensure the system design can accommodate future high frequency modules. Intel has provided the maximum possible bulk capacitance on the module. However, in order to achieve proper filtering and in-rush current protection, it is imperative that additional filtering be provided on the system electronics board. Table 1 details the bulk capacitance requirements for the system electronics. Also, take special note of the voltage rating requirement for the capacitors on each respective voltage rail.

Note that the rating requirement and note 5 has been changed from previous versions of this table in the *Celeron Processor Mobile Module datasheet*. Future revisions of this specification will include these new changes.

Table 1. Bulk Capacitance

Power Plane	Capacitance Requirements	Max Total ESR	Ripple Current	Rating
V_DC	100 uf [5], 0.1 uf, 0.01 uf[1]	20 mΩ	4.5A-7.5A[3] 3A ~ 5A[4]	20% tolerance at 35V
V_5	100 uf[5], 0.1 uf, 0.01 uf[1]	100 mΩ	1A	20% tolerance at 10V
V_3	470 uf[5], 0.1 uf, 0.01 uf[1]	100 mΩ	1A	20% tolerance at 6V
V_3S	100 uf[5], 0.1 uf, 0.01 uf[1]	100 mΩ	N/A	20% tolerance at 6V
VCC_AGP	22 uf[5], 0.1 uf, 0.01 uf[1]	100 mΩ	1A	20% tolerance at 6V
V_CPUPU	2.2 uf, 8200 pf[1]	N/A	N/A	20% tolerance at 6V
V_CLK	10 uf[5], 8200 pf[2]	N/A	N/A	20% tolerance at 6V

NOTES:

1. Placement of above capacitance requirements should be located near the connector.
2. V_CLK filtering should be located next to the system clock synthesizer.
3. Ripple current specification depends on V_DC input for the Geyserville module.
4. Ripple current specification depends on V_DC input for the single frequency module.
5. If Tantalum Capacitors are used, a 50% voltage de-rating practice must be observed (for example, a 5.0-V rail requires a 10.0-V rated capacitor).

SPECIFICATION CHANGES

The Specification Changes listed in this section apply to the following documents:

- *P6 Family of Processors Hardware Developer's Manual*
- *Intel® Mobile Celeron® Processor in Micro-PGA and BGA Package at 466 MHz, 433 MHz, 400 MHz, 366 MHz, 333 MHz, 300 MHz and 266 MHz* datasheet
- *Intel® Celeron® Processor Mobile Module: Mobile Module Connector 1 (MMC-1) at 400 MHz, 366 MHz, 333 MHz, 300 MHz, and 266 MHz* datasheet
- *Intel® Celeron® Processor Mobile Module: Mobile Module Connector 2 (MMC-2) at 400 MHz, 366 MHz, 333 MHz, 300 MHz, and 266 MHz* datasheet
- *Intel® Mobile Celeron® Processor: Mobile Module MMC-1 at 466 MHz and 433 MHz* datasheet
- *Intel® Mobile Celeron® Processor: Mobile Module MMC-2 at 466 MHz and 433 MHz* datasheet
- *Intel Architecture Software Developer's Manual, Volumes 1, 2 and 3*

All Specification Changes will be incorporated into a future version of the appropriate Mobile Intel Celeron processor documentation.

H1. $I_{CC,SG}$, $I_{CC,QS}$, and $I_{CC,DSLP}$ Maximum Specifications for BGA1 and Mini-Cartridge

In *the Intel® Mobile Celeron® Processor in BGA Package at 300 MHz and 266 MHz* datasheet, maximum $I_{CC,SG}$, $I_{CC,QS}$, and $I_{CC,DSLP}$ specifications have been increased by 250 mA, to 1190 mA, 880 mA and 650 mA respectively. The following is the updated Table 3.6 of the datasheets.

Table 3.6. Mobile Celeron® Processor Power Specifications[1]

T_{CASE} = 0 to $T_{CASE,max}$; V_{CC} = 1.6 V ±135 mV; V_{CCP} = 1.8 V ±90 mV						
Symbol	**Parameter**	**Min**	**Typ**	**Max**	**Unit**	**Notes**
V_{CC}	V_{CC} of core logic for regular voltage processors	1.465	1.6	1.735	V	±135 mV
$V_{CC,LP}$	V_{CC} when I_{CC} < 300 mA	1.465	1.6	1.805	V	+205/-135 mV [2]
V_{CCP}	V_{CC} for CMOS voltage references	1.71	1.8	1.89	V	1.8 V ±90 mV
I_{CC}	I_{CC} for V_{CC} at core frequency @ 300 MHz @ 266 MHz			7.49 6.63	A A	5
I_{CCP}	Current for V_{CCP}			75	mA	3, 4, 5
$I_{CC,SG}$	Processor Stop Grant and Auto Halt current			1190	mA	5
$I_{CC,QS}$	Processor Quick Start and Sleep current			880	mA	5

Table 3.6. Mobile Celeron® Processor Power Specifications[1]

T_{CASE} = 0 to $T_{CASE,max}$; V_{CC} = 1.6 V ±135 mV; V_{CCP} = 1.8 V ±90 mV						
Symbol	**Parameter**	**Min**	**Typ**	**Max**	**Unit**	**Notes**
$I_{CC,DSLP}$	Processor Deep Sleep leakage current			650	mA	5
dI_{CC}/dt	V_{CC} power supply current slew rate			20	A/µs	6, 7

NOTES:

1. Unless otherwise noted, all specifications in this table apply to all processor frequencies.
2. A higher $V_{CC,MAX}$ is allowed when the processor is in a low power state to enable high efficiency, low current modes in the power regulator.
3. I_{CCP} is the current supply for the CMOS voltage references.
4. Not 100% tested. Specified by design/characterization.
5. $I_{CCx,max}$ specifications are specified at $V_{CC,max}$, $V_{CCP,max}$ and 100° C and under maximum signal loading conditions. $I_{CCx,max}$ specifications are not specified at $V_{CC,LP,max,}$ if that voltage specification is used then slightly higher currents can be expected.
6. Based on simulations and averaged over the duration of any change in current. Use to compute the maximum inductance and reaction time of the voltage regulator. This parameter is not tested.
7. Maximum values specified by design/characterization at nominal V_{CC} and V_{CCP}.

H2. Temperature Note For Thermal Power Specifications

In the *Intel® Mobile Celeron® Processor in BGA Package at 333 MHz, 300 MHz and 266 MHz* datasheet, Table 6.1 TDP notes should state “at 100° C[2, 3]” instead of “at 50° C[2, 3]”. Table Note 3 should state, “3. Not 100% tested or guaranteed. The power specifications are composed of the current of the processor on the various voltage planes. These currents are measured and specified at high temperature in Section 3.5. The 50° C power specifications are determined by characterization of the processor currents at higher temperatures.”